SPECIALIZATIONELECTIVE

DATA SCIENCE (22DS102006)

Module 1 INTRODUCTION

1. Definition of data science

Data Science (DS): Study and practice of collecting, storing, processing, and analyzing data to extract meaningful insights and support decision-making. It includes scientific methods, computational thinking, and systematic analysis of data. DS combines computer science, statistics, mathematics, and domain expertise.

The 3Vs of data that justify its rise:

- Volume**:** Massive amounts of data (e.g., zettabytes)
- Velocity**:** Speed at which data is generated
- Variety**:** Structured and unstructured data formats (text, image, video, etc.)

Data science is used to derive meaningful insights to solve real-world problems through systematic, repeatable, and verifiable processes.

Figure 1.1 shows that, by 2020, global data usage would increase 40 zettabytes—a 50-fold increase since 2010. This proves increasing importance of DS in managing and analyzing this massive amount of information.

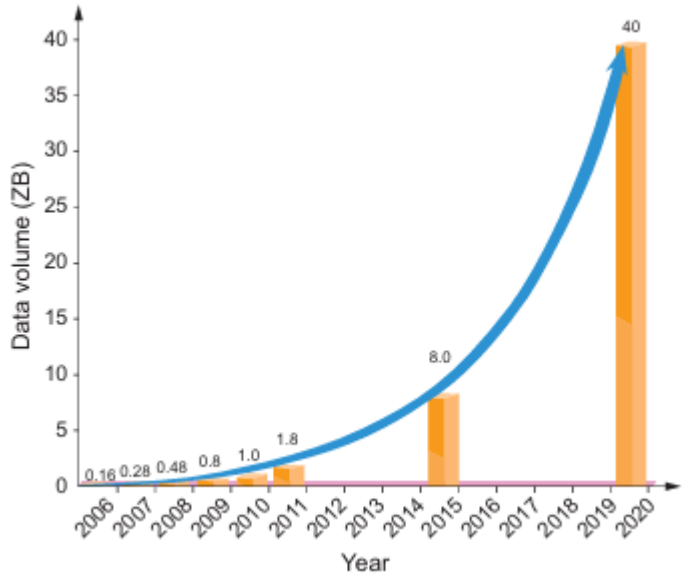


Figure 1.1: Increase of Data Volume in 15 Years

(Source: IDCs Digital Universe Study, December 2012)

Where Do We See Data Science?

Data science finds widespread application across a multitude of domains owing to its capability to extract meaningful insights from vast and complex datasets. Below are some key areas where data science plays a transformative role:

- **Finance**: In the financial sector, data science facilitates fraud detection, credit risk assessment, customer segmentation, and the formulation of investment strategies. By analyzing transactional data and historical customer records, predictive models can be developed to support decisions such as loan approvals and portfolio management.
- **Public Policy**: Governments leverage data science to examine citizen behavior, traffic flow, usage of public services, and social trends, thereby enhancing governance and infrastructure development. Open data platforms, such as data.gov, provide access to datasets that support evidence-based policymaking across diverse areas including sanitation, transportation, and urban planning.
- **Politics**: Political campaigns increasingly rely on data science for voter profiling, targeted campaigning, and sentiment analysis via social media. By identifying undecided voters and customizing campaign messages, political strategists can optimize outreach and engagement efforts.
- **Healthcare**: In the healthcare industry, data science revolutionizes disease diagnosis, patient data management, and personalized treatment plans. Wearable health devices, such as fitness trackers, continuously collect physiological data, which can be analyzed to alert users and physicians about potential health risks and support preventive care.
- **Urban Planning**: Smart city projects benefit significantly from data science through the integration of sensor networks and analytical tools to optimize traffic control, energy consumption, and public safety. Examples include systems like Chicago's Plow Tracker and Boston's SnowCOP, which utilize real-time data to enhance municipal service delivery.
- **Education**: Educational institutions use data science to support personalized learning by monitoring student performance and engagement. These insights enable educators to tailor teaching strategies and deliver timely academic interventions.
- **Libraries and Information Science**: Libraries apply data science to automate systematic literature reviews, enhance metadata management, and streamline access to academic resources. These capabilities assist researchers in identifying relevant materials more efficiently.
- **Business Sector**: In business, data science strengthens marketing, sales, and customer support by analyzing consumer behavior, forecasting product demand, and delivering personalized recommendations. Tech giants like Amazon and Netflix utilize sophisticated data models to optimize user experience and drive customer retention.
- **Industrial Sectors**: Manufacturing and industrial operations use data science for predictive maintenance, supply chain optimization, and quality assurance. With the rise of IoT devices, real-time data collection from machinery enables smarter production and process control.

Across these diverse fields, data science stands out as a powerful enabler for uncovering patterns, making accurate predictions, and facilitating data-driven decisions.

2. Skills for data science

The success of DS involves adopting a mindset that merges curiosity, analytical thinking, and technical expertise. Three key skills used in DS are willingness to experiment, mathematical reasoning, and data literacy. These skills reflect not only what employers seek but also what enables professionals to succeed in this evolving, interdisciplinary domain. The hybrid nature of a data scientist, need the following essential skills:

a) Willingness to Experiment

- Curious, creative problem-solvers who define hypotheses and explore data.
- Employers often assess curiosity through open-ended questions.

b) Proficiency in Mathematical Reasoning

- Solid understanding of statistics and logical reasoning.
- Interpretation of numeric data is critical for analysis.

c) Data Literacy

- The ability to read, interpret, and work with data.
- Involves understanding data relevance, visualizing insights, and making data-driven decisions.
- Considered a fundamental 21st-century skill, like reading and writing.

d) Roles and Job Types in Data Science

- Data Analyst: Entry-level role focused on visualizations, reporting, Excel, Google Analytics, etc.
- Data Engineer: Designs infrastructure for handling large volumes of data.
- Product-Focused Data Scientist: Develops data-driven services/products (e.g., recommendation engines).
- Generalist in Data-Driven Companies: Wears multiple hats including ML, visualization, reporting.

In another view, Dave Holtz blogs about specific skill sets desired by various positions to which a data scientist may apply

A) Data Analyst as Entry-Level Data Scientist: In many companies, entry-level data scientists perform tasks similar to data analysts—using tools like Excel, MySQL, Google Analytics, and Tableau for data retrieval, reporting, and visualization. These roles focus on descriptive analysis and serve as a practical starting point for beginners.

B) Data Wrangler / Engineer Roles: Some organizations hire data scientists to manage large, unstructured datasets. These roles, often labeled as data engineers, involve building infrastructure and data pipelines. They require strong programming skills and are ideal for self-motivated individuals, though they often lack formal mentorship.

C) Data-Product Focused Roles: In data-centric companies, data scientists develop products powered by data—such as recommendation engines or predictive models. These roles emphasize machine learning and advanced analytics, typically attracting candidates with academic backgrounds in math, statistics, or physics.

D) Generalist Roles in Data-Driven Companies: Many businesses use data to support decision-making. Data scientists here work on established teams and require broad skills—analysis, visualization, coding—along with specialization in areas like ML or dashboard design. Familiarity with big data tools and messy datasets is often essential.

The above-discussed skills are summarized in Figure 1.2.

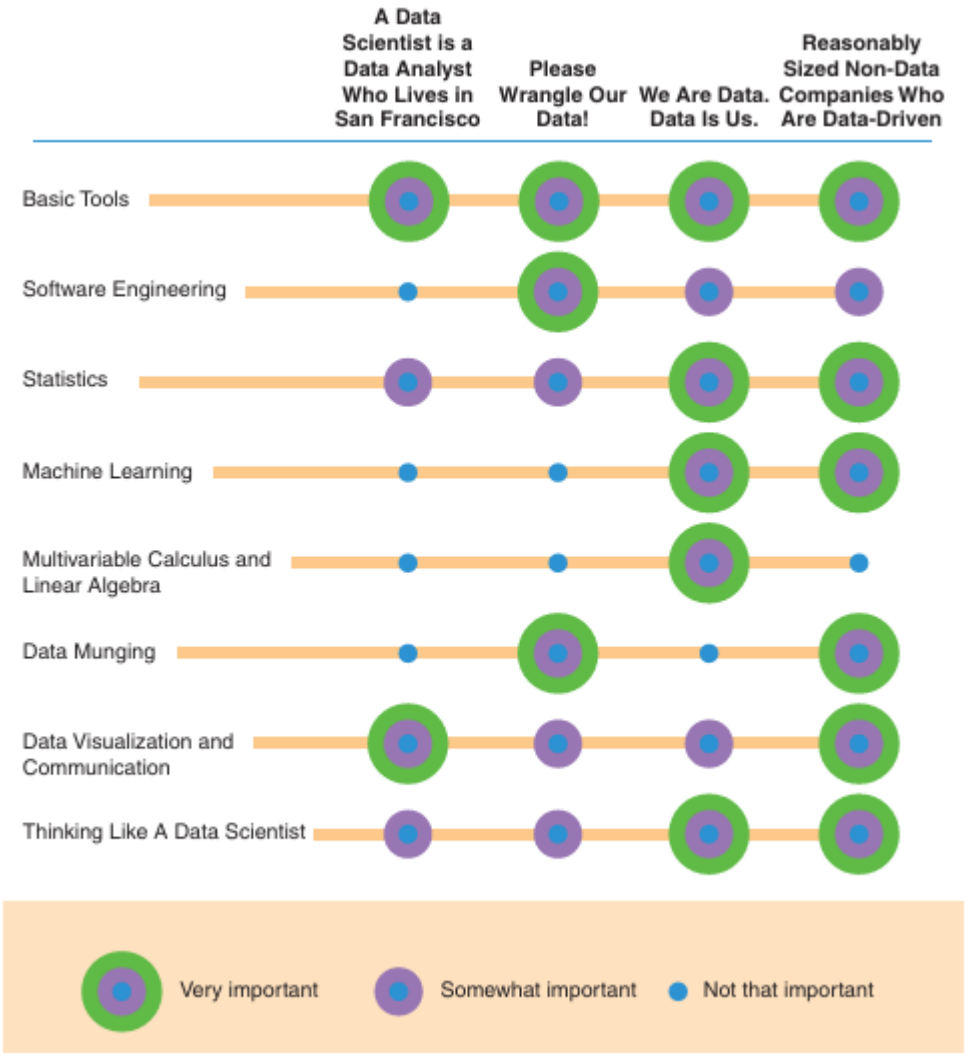


Figure 1.2: Types of Data Science Roles

3. Tools for data science

Becoming a successful data scientist requires hands-on experience with tools that are widely used in the industry for solving data-driven problems. They offer powerful capabilities for implementing, visualizing, and testing data solutions quickly. Python and R, on the other hand, offer user-friendly environments with extensive libraries that facilitate easy setup and enable beginners to produce meaningful results.

A) Python

- High-level scripting language
- Interpreted line by line (no need for compiling like Java/C++)
- Simple syntax; easy to learn and write

- Widely supported with open-source libraries and packages
- Suitable for both beginners and advanced users

Common Use Cases:

  - Data manipulation: Pandas
  - Numerical computing: NumPy, SciPy
  - Visualization: Matplotlib, Seaborn
  - Machine learning: scikit-learn, TensorFlow, PyTorch

B) R

- Statistical programming language
- Designed specifically for statistical analysis and data visualization
- Simple syntax similar to natural mathematical expressions
- Strong community support for packages in CRAN (Comprehensive R Archive Network)

Common Use Cases:

  - Statistical modeling
  - Hypothesis testing
  - Data visualization (with ggplot2)
  - Report generation using RMarkdown

C) SQL (Structured Query Language)

- Domain-specific language for managing relational databases
- Used for querying and manipulating structured data
- Efficient for handling large datasets stored in relational databases
- Can be used within Python or R for integration with large-scale databases

Common Use Cases:

  - Extracting specific data subsets
  - Filtering, joining, and aggregating data tables
  - Working with real-time or remote datasets

SQL is particularly useful when data is too large to load into memory as a flat file (like CSV).

D) UNIX (Shell/Command Line)

- Operating system environment and set of command-line tools
- Supports automation and batch data processing
- Powerful for file handling, text processing, and scripting
- Reduces the need for coding in certain routine data tasks

Common Use Cases:

  - Filtering and processing logs
  - Running scripts and pipelines
  - Managing large datasets without GUI

UNIX is often used in combination with Python or R in professional environments, especially on servers or cloud-based systems.

Need of These Tools

These tools are popular not because they were built specifically for data science, but because:

- They support rapid development and iteration
- They are open-source and backed by strong community ecosystems
- They interoperate well—e.g., using SQL with Python, or R with UNIX
- They handle a wide variety of data formats and volumes

Other Mentions

While the file mainly emphasizes Python, R, SQL, and UNIX, in practice, other tools that data scientists often encounter include:

- Jupyter Notebooks: For interactive Python code + narrative
- Tableau / Power BI: For dashboarding and business intelligence (BI)
- Hadoop / Spark: For distributed data processing (Big Data)

4. Data Types

One of the most basic concepts in data The most practical classification of data would be to classify it as either structured or unstructured. This difference is of high importance since methods of storage, querying, processing, and analysis are frequently differentiated according to this division. The structured data is very much organized and it is mostly available in tabular form with labeled rows and columns and it can be handled comfortably with the help of databases. On a different note, unstructured data is not pre-structured and could be in the form of the free text, audio, video or images. Most of the world data is unstructured, therefore handling it tends to be more complex and requires special processing procedures.

Structured Data

Structured data: this can also be called neatly organized data most often in a tabular format with fields. In this structure, each data point is linked to a label or any particular attribute in which case, it is easy to search, analyze and visualize. Consider, as an example, values of height and weight of people; in this case, a dataset is called structured, since every numeric value has a label, which explains its purpose. Structured data doesn't always have to be numerical; it can include other types such as text (e.g., "housing type"), Boolean (e.g., "is employed"), or categorical variables (e.g., "gender" or "marital status"). The exact format of values does not determine structured data; it is the definite structure of naming and labelling.

To illustrate this, a table (1.1) might contain records with a mix of data types—such as an individual's age, employment status, and place of residence.

Table 1.1: Customer Data Sample

custid	sex	is.employed	income	marital.stat	housing.type	num.vehicles	age	state.of.res
2068	F	NA	11300	Married	Homeowner free and clear	2	49	Michigan
2073	F	NA	0	Married	Rented	3	40	Florida
2848	M	True	4500	Never married	Rented	3	22	Georgia
5641	M	True	20000	Never married	Occupied with no rent	0	22	New Mexico
6369	F	True	12000	Never married	Rented	1	31	Florida

Unstructured Data

On the contrary, unstructured data is disorganized and not labeled. It is not tied to a certain model or format, therefore, it is more difficult to manage and interpret it. The unstructured form of data would present a good example of a narrative paragraph in a medical or social study. Such an example can be the following combination of words: "It was discovered that a woman measured between 65 and 67 inches and measured 125130 IQ points." Although this has valuable information, it is not written in a way that makes it easily parsed by software. It does not contain column names or immediate labels that interrelate any piece of data to a unique entity or property. Consequently, interpreting and processing this type of data may be a task that needs human revising or the usage of specialized natural language processing software.

The major challenge of unstructured data is represented by its ambiguity. It can either be context-related or a semantic interpretation meaning that it is even harder to automate. In contrast to structured data, one cannot be sure of the meaning of the term 22 in labelled column, when it comes to unstructured data, one has to follow more paths to come out of similar meanings. Nevertheless, unstructured data is an untapped pool of knowledge, especially when used in the social sphere, customer notes, or multimedia entries, but with the correct methodologies to process and organize it, data is still information that can provide excellent insight.

5. Data collections

When looking for datasets similar to those discussed in earlier sections or chapters, there are several reliable online sources available. These resources provide access to various data formats for analysis, research, and application development.

5.1. Open Data

Open data refers to datasets that are made freely accessible to the public without legal or licensing restrictions. The goal is to allow unrestricted usage, modification, and distribution. Governments (both local and national), NGOs, and academic institutions are major contributors to open data initiatives. For instance, datasets are published by the U.S. Government and cities like Chicago for public use.

In 2013, the White House launched Project Open Data to promote the open data policy by offering resources such as tools, case studies, and guidelines. A policy memo (M-13-3) was also introduced, emphasizing the need to treat data as a valuable asset and release it in accessible formats whenever possible.

Key Principles of Open Data:

Public: Agencies are encouraged to share data openly, unless restricted by privacy, security, or legal concerns.

Accessible: Data must be provided in user-friendly, modifiable formats that support search and download. These formats should ideally be machine-readable and non-proprietary.

Described: Open data must include sufficient metadata and documentation to help users understand the data's structure, limitations, and collection methods.

Reusable: Data should be shared under an open license, with no limitations on reuse.

Complete: Wherever possible, data should be released in its raw, original form, along with any derived versions.

Timely: Data should be made available promptly to retain its relevance and usefulness.

Managed After Release: A designated contact should be available to support data users and respond to compliance issues.

5.2. Social Media Data

Social media platforms are valuable sources for collecting user-generated content for research and marketing. These platforms offer APIs (Application Programming Interfaces) that enable developers and researchers to programmatically retrieve information such as user profiles, posts, and engagement metrics.

An example is the Facebook Graph API, which allows data extraction in structured formats like XML. These APIs enable developers to create innovative applications, conduct behavioral studies, or monitor responses to events like natural disasters. Some platforms, like Yelp, actively release datasets to encourage exploration and academic research. These datasets have supported projects in areas like photo classification, natural language processing (NLP), sentiment analysis, and network analysis. Researchers can participate in initiatives like the Yelp Dataset Challenge to solve real-world data problems.

5.3. Multimodal Data

In today's increasingly connected world, many devices—from smart bulbs to vehicles—generate diverse types of data as part of the Internet of Things (IoT) ecosystem. This data often goes beyond traditional formats (like numbers or text) and includes rich content such as images, audio, gestures, and spatial data.

Data from such sources is known as multimodal or multimedia data, and it often requires specialized techniques to analyze. For example, in medical research, brain imaging data can come from multiple sensors like EEG, MEG, and fMRI. These measurements, captured in sequences or time-series, are analyzed using techniques like Statistical Parametric Mapping (SPM)—a method developed by Karl Friston—to identify patterns of brain activity.

For those interested in working with such complex datasets, Appendix E (not included here) offers guidance on additional sources and ongoing challenges related to data analysis and real-world problem solving.

5.4. Data Storage and Presentation

Data can be stored in various formats depending on its structure and intended use. The following are common formats for storing and presenting data:

A) CSV (Comma-Separated Values)

- Widely adopted for sharing spreadsheets and database exports.
- Stores data in plain text with fields separated by commas.

Advantages: Universally readable by spreadsheet tools like Excel and Google Sheets; easy to share and understand.

Challenges: If the data itself contains commas, parsing becomes difficult unless proper escape characters are used.

Example: A file named Depression.csv from UF Biostatistics contains records of treatments and outcomes for individuals with depression.

B) TSV (Tab-Separated Values)

- Often used to store raw data between systems or tools.
- Format: Similar to CSV, but fields are separated by tabs instead of commas.

Advantages: Reduces ambiguity since tabs are less likely to occur in the data itself.

Challenges: Less commonly used than CSV; requires attention when tabs exist within field content.

Example: Employee registration data stored with fields like Name, Age, and Address separated by tab spaces.

C) XML (eXtensible Markup Language)

- Designed to be both human-readable and machine-readable.
- Format: Structured text using custom tags (e.g., <book>, <price>).

Advantages: Platform-independent and suitable for data exchange between incompatible systems.

Real-world Relevance: Many IT systems use XML to standardize communication between databases and applications. XML is also familiar to those who have worked with HTML.

D) RSS (Really Simple Syndication)

- RSS is a web-based format used to distribute content between platforms and was introduced with version 1.0 of XML. It allows websites to deliver updated information, like news or blog posts, in a standardized XML file known as an RSS feed. While most modern web browsers can read RSS feeds directly, users often rely on RSS readers or aggregators, which are tools designed to collect and organize feeds from multiple sources.
- RSS uses standard XML syntax but extends it by specifying particular tags (some mandatory, others optional) and the type of content each tag should contain. It is tailored for delivering selective, structured content efficiently.

Example Use Case**:**
Imagine running a website that publishes frequent updates such as news headlines, stock

prices, or weather alerts. Users interested in staying informed would need to check the site repeatedly, which can be inefficient—either they miss updates or waste time checking unnecessarily. Using an RSS feed, users can subscribe through an aggregator, which automatically checks for updates and delivers them as soon as they're available—often through notifications.

Why RSS is Useful**:**

- It is lightweight and quick to load.
- Easily integrates with devices like smartphones, smartwatches, and PDAs.
- Ideal for frequently updated platforms such as:
    - News websites (providing headlines, dates, and brief summaries)
    - Business pages (announcements and new product updates)
    - Calendars (event reminders and important dates)
    - Websites with frequent content changes (tracking new or modified pages)

E) RSS (Really Simple Syndication)

- RSS is a web-based format used to distribute content between platforms and was introduced with version 1.0 of XML. It allows websites to deliver updated information, like news or blog posts, in a standardized XML file known as an RSS feed. While most modern web browsers can read RSS feeds directly, users often rely on RSS readers or aggregators, which are tools designed to collect and organize feeds from multiple sources.
- RSS uses standard XML syntax but extends it by specifying particular tags (some mandatory, others optional) and the type of content each tag should contain. It is tailored for delivering selective, structured content efficiently.

Example Use Case:

Imagine running a website that publishes frequent updates such as news headlines, stock prices, or weather alerts. Users interested in staying informed would need to check the site repeatedly, which can be inefficient—either they miss updates or waste time checking unnecessarily. Using an RSS feed, users can subscribe through an aggregator, which automatically checks for updates and delivers them as soon as they're available—often through notifications.

Why RSS is Useful:

- It is lightweight and quick to load.
- Easily integrates with devices like smartphones, smartwatches, and PDAs.
- Ideal for frequently updated platforms such as:
- News websites (providing headlines, dates, and brief summaries)
- Business pages (announcements and new product updates)
- Calendars (event reminders and important dates)
- Websites with frequent content changes (tracking new or modified pages)

6. Data Preprocessing

In the real world, data is rarely perfect. It often needs to be cleaned, reformatted, or adjusted before it can be analyzed effectively. This entire process is referred to as data preprocessing, and it is a critical first step in any data analysis or machine learning workflow.

Common Problems in Raw Data

Data is considered "dirty" when it contains any of the following issues:

- Incomplete Data**:** Missing attribute values or records that only provide partial information.
- Noisy Data**:** Data with errors or outliers that distort the actual trend.
- Inconsistent Data**:** Mismatched formats, naming discrepancies, or improperly formatted entries.

Major Preprocessing Tasks

A) Data Cleaning

Data cleaning focuses on identifying and correcting issues in raw data. It may involve:

- Fixing incorrect or inconsistent values
- Removing duplicates
- Handling missing values
- Standardizing text entries (e.g., names or categories)

B) Data Munging (Wrangling)

Data munging involves reshaping or converting data into a usable format. This may be necessary when data is stored in a form that's difficult to analyze—like a paragraph or an unstructured text. There is no single method for wrangling; it often requires creative or manual solutions to make data "analysis-friendly."

C) Handling Missing Values

Missing data can occur due to user input errors, device malfunctions, or insufficient data collection at the time. Strategies to handle missing values include:

- Dropping records
- Filling with a default value or average
- Using statistical methods like inference or imputation

D) Smoothing Noisy Data

Noisy data—caused by errors in data entry, sensor faults, or loss of precision—can be reduced by:

- Detecting and removing outliers
- Standardizing data formats
- Applying smoothing techniques (e.g., averaging)

E) Data Integration

When data comes from multiple sources, integration is necessary. This involves:

- Merging databases or files into a unified dataset
- Resolving schema and attribute conflicts
- Identifying and eliminating redundant data
  Example: Combining revenue data from different systems using different units (e.g., INR vs. USD)

F) Data Reduction

Data reduction helps simplify datasets without losing essential information. Two main techniques:

- Aggregation**:** Condensing detailed records into summarized forms (e.g., monthly totals from daily sales)
- Dimensionality Reduction**:** Removing irrelevant or redundant features using methods like PCA, clustering, or feature merging

G) Data Discretization

Discretization involves converting continuous data (like temperature or stock prices) into categories or ranges. This simplifies analysis, especially for numerical attributes.

- Types of attributes:
  - Nominal**:** No inherent order (e.g., color)
  - Ordinal**:** Ordered categories (e.g., education level)
  - Continuous: Numeric values (e.g., age, temperature)
- Example: Grouping temperature into "cold", "moderate", and "hot"

7. Data analysis and data analytics

Although often used interchangeably, the terms data analysis and data analytics are not exactly the same. This distinction, while subtle, is significant—especially for professionals working with data. Misinterpreting the terms may hinder effective use of data for insights and decision-making.

Dave Kasik, a Senior Technical Fellow at Boeing, offers a helpful perspective. In his view, data analysis refers to direct, hands-on work involving the exploration and assessment of data. On the other hand, data analytics encompasses a broader scope, with data analysis being just one of its components. Kasik describes analytics as the science behind the analytical process, focusing on how analysts cognitively approach and solve data problems.

A practical way to differentiate them is through a time-based lens:

- Data analysis generally deals with understanding what has already happened. It provides a retrospective view, commonly used in fields like marketing to evaluate past performance.
- Data analytics looks forward—it leverages models and statistical tools to forecast outcomes, offering insights that guide future actions and strategies.

Analytics heavily relies on mathematical and statistical techniques. It includes not just interpreting data, but also using descriptive and predictive models to extract valuable insights. These insights are then used to support business decisions or recommend courses of action. Rather than focusing solely

on individual analysis tasks, analytics considers the entire framework and methodology behind data-driven problem-solving.

Although there's no universally accepted classification of data science techniques, a practical approach is to group them based on their purpose and stage in the data lifecycle. Accordingly, the following six categories can be used to classify common analytical techniques:

- Descriptive Analysis
- Diagnostic Analytics
- Predictive Analytics
- Prescriptive Analytics
- Exploratory Analysis
- Mechanistic Analysis

Each of these plays a distinct role in how data is interpreted and applied in real-world scenarios.

8. Descriptive analysis

Descriptive analysis focuses on summarizing current data to understand “what is happening now.” It provides a quantitative overview of the main characteristics of a dataset and is typically the initial step in data examination.

Key Features

- First Step: Often the first analysis done on large datasets like census information.
- Volume-Ready: Well-suited for large-scale data.
- Two-Phase Process: Data description precedes interpretation.

Descriptive statistics are essential for organizing raw data into a structured form that reveals patterns. These summaries help identify insights, but accurate interpretation demands appropriate statistical methods.

Why Use Descriptive Analysis?

- Helps group customers by preferences or behavior.
- Enables population-wide summaries (e.g., census).
- Transforms raw data into meaningful insights.

8.1. Variables

Variables are labels assigned to the data being studied. For example, a column labeled "Age" in a spreadsheet represents a numeric variable. Variables are categorized as:

- Nominal: Represent categories without order (e.g., colors).
- Ordinal: Categories with inherent order (e.g., rankings).
- Interval: Numeric values with meaningful differences, but no true zero (e.g., temperature in Celsius).
- Ratio: Numeric values with a true zero, allowing all arithmetic operations (e.g., weight, height).

Variables are also classified based on their role:

- Independent/Predictor: The cause or influencing factor.
- Dependent/Outcome: The effect or the result we are predicting.

8.2. Frequency Distribution

A frequency distribution shows how often each value appears. It can be visualized using:

- Histogram: Bars show frequency of numerical values.
- Pie Chart: Useful for visualizing categorical data proportions.

Example:

A histogram of productivity scores can quickly show the most frequent ranges of performance in a workforce.

8.3. Normal Distribution

An ideal dataset follows a bell-shaped curve:

- Symmetrical Center: Most values near the mean.
- Skewness: Indicates asymmetry (positive or negative).
- Kurtosis: Measures the "peakedness" or flatness of the distribution.

Visual tools like histograms help assess normality.

8.4. Measures of Centrality

These metrics summarize the center of a dataset:

- Mean: Arithmetic average. Sensitive to outliers.
- Median: Middle value when data is sorted. Better for skewed data.
- Mode: Most frequently occurring value. Best for categorical data.

Each measure provides a different lens on the dataset and choosing the right one depends on the data type and distribution.

8.5. Dispersion of Distribution

Dispersion tells us how spread out the data is:

- Range: Difference between the largest and smallest values.
- Interquartile Range (IQR): Range of the middle 50%, useful for reducing outlier influence.
- Variance: It is a statistical measure that indicates how much the values in a dataset differ from the average (mean). It reflects the degree to which data points are spread out. The more the values deviate from the mean, the higher the variance; conversely, closely grouped values result in a lower variance.There are two types of variance:

    A) Population Variance (σ^2)
    It measures the dispersion of the entire population. The formula is:

$$\sigma^2 = \frac{\sum(X_i - \overline{X})^2}{N}$$

where:

- σ^2 = population variance
- X_i = individual data value
- $\overline{X}$ = population mean
- N = total number of values in the population

B) Sample Variance (s^2)
It measures the spread within a sample subset of the population and adjusts the denominator to avoid bias. The formula is:

$$s^2 = \frac{\sum(x_i - \overline{x})^2}{n - 1}$$

where:

- s^2 = sample variance
- x_i = individual sample value
- $\overline{x}$ = sample mean
- n = sample size

8.6. Standard Deviation

Although variance measures spread, it presents the result in squared units (e.g., years^2), which is not intuitive. To convert it back into the original unit of measurement, we take the square root of the variance, resulting in the standard deviation.

The formula for sample standard deviation is:

$$s = \sqrt{\frac{\sum(x_i - \overline{x})^2}{n - 1}}$$

Standard deviation gives a clearer idea of the average distance of each data point from the mean, expressed in the same unit as the original data (e.g., years, dollars, etc.).

9. Diagnostic analytics

Definition:
Diagnostic analytics focuses on understanding *why* an event occurred. It investigates the causes behind trends or anomalies identified in descriptive analytics.

Key Characteristics:

- Also known as causal analysis.
- Explores relationships between causes and outcomes.
- Useful for reviewing past performance and identifying contributing factors.

Example Use Case:
In a social media campaign, after using descriptive analytics to track metrics (likes, comments, shares), diagnostic analytics helps understand *why* certain posts performed better than others.

Common Technique – Correlation:

- Measures how strongly two variables are related.
- Value ranges from -1 (strong negative) to +1 (strong positive).

10. Predictive analytics

Definition:
Predictive analytics estimates what might happen in the future by analyzing patterns in current and historical data.

Key Concepts:

- Based on probabilities, not certainties.
- Generates foresight from hindsight and insights.
- Often uses regression to confirm data relationships.

Four-Step Process:

- Data Collection – Gather historical and current data.
- Data Cleaning – Ensure data quality for meaningful analysis.
- Pattern Recognition – Use visualizations and statistical models.
- Prediction – Apply models to forecast outcomes.

Applications:

- Marketing (forecasting sales)
- Healthcare (predicting patient risks)
- Finance (credit scoring, fraud detection)

Tools Used: SAS, IBM SPSS, RapidMiner, etc.
Example: Estimating future sales based on historical advertising spend across platforms

11. Prescriptive analytics

Definition:
Prescriptive analytics identifies the best course of action from available options. It goes beyond predictions to recommend decisions.

How It Works:

- Starts with descriptive/predictive insights.
- Evaluates possible decisions and their impact.
- Suggests optimal strategies using simulations, game theory, and optimization models.

Use Case in Healthcare:
Analyzing obese patient data along with diabetes and cholesterol levels to determine priority treatment zones.

Challenges:

- Computationally intensive.
- Still underutilized in industries (used by only ~3% of organizations).

Benefits:

- Provides clear decision paths.
- Improves proactive planning.

12. Exploratory analysis

Definition:
Exploratory Data Analysis (EDA) is used when the problem is not clearly defined. It helps uncover hidden patterns and relationships in data.

Approach:

- Involves visualizations (scatterplots, bar charts).
- Encourages open-ended exploration without hypotheses.
- Acts as a starting point for more formal analysis.

Philosophy:
EDA is more about mindset than tools—it guides how to examine data rather than what techniques to use.

Application Example:
Using US Census data to search for unexpected trends across dozens of variables before deciding on specific analytical methods.

Limitations:

- Should not be used for drawing conclusions or predictions on its own.
- Serves to inform further analysis rather than end it.

13. Mechanistic analysis.

Definition:
Mechanistic analysis investigates how specific changes in one or more variables bring about changes in others, often in physical or controlled systems.

Use Case Examples:

- Studying the effect of CO_2 levels on global temperature changes.
- Assessing how increasing employee perks affects productivity.

Core Technique – Regression Analysis:

- Establishes mathematical relationships between variables.
- Helps predict outcomes (e.g., score = f(attitude)) using linear models:

$$y = \beta_0 + \beta_1 x$$

Where:

- β_0 = Intercept (baseline),
- β_1 = Slope (effect of x on y),
- x = Independent variable (predictor),
- y = Dependent variable (outcome)

When to Use:

- To model cause-effect relationships at an individual object level.
- Especially suitable for scientific and industrial applications.

MODULE 2

DATA EXTRACTION

1. Extracting meaning from data

Companies extract meaning from the data from two people with very different approaches namely, William Cukierski from Kaggle and David Huffaker from Google.

A) William Cukierski - Kaggle

William Cukierski underscores the critical role of both feature extraction and feature selection in the field of data science. He presents the following key ideas:

- Feature Extraction**:** This process involves converting raw data into a well-structured and clean format, thereby preventing the common issue of unreliable outputs caused by poor input data. The goal is to ensure the dataset used in modeling is both relevant and refined.
- Feature Selection**:** After extraction, the next step is to isolate a subset of data attributes—or combinations of them—that are most useful for modeling. This step is essential for reducing redundancy and enhancing model performance by concentrating only on significant variables.
- Kaggle Challenges**:** Kaggle offers a platform for global data science competitions. Participants receive training and test datasets to build predictive models. These events foster continuous innovation, where participants build upon each other's progress in a cycle of improvement.
- Crowdsourced Problem Solving**:** By tapping into a worldwide network of data scientists, organizations can access a wide range of ideas and expertise. This collective approach often leads to more creative and effective solutions to complex analytical problems.

Kaggle Competition Framework:

Kaggle is a popular online platform that transforms data science challenges into competitive events. Organizations collaborate with Kaggle to host data-driven contests, inviting data scientists from around the globe to develop solutions. Participants receive both training and testing datasets; however, the test set contains concealed target values. Competitors build models to predict these hidden values and submit their outputs to Kaggle, which ranks them on a dynamic leaderboard.

These competitions encourage ongoing refinement of models, as frequent submissions often lead to successive improvements. This iterative process creates a phenomenon where participants continuously outperform each other, commonly known as the "leapfrogging" effect. Nevertheless, extended competitions may lead to unnecessarily complex model architectures, as observed during the Netflix Prize contest.

Understanding Leapfrogging:

The concept of leapfrogging refers to bypassing intermediate stages of progress or development in favor of more advanced approaches. This can be categorized into several contexts:

- **Economic Leapfrogging:** Emerging nations often avoid legacy technologies and transition directly to modern systems. For instance, instead of investing in landline infrastructure, many countries adopted mobile phone networks, resulting in faster and more cost-effective connectivity.
- **Technological Leapfrogging:** Within the corporate sector, some firms skip gradual technological upgrades and instead implement cutting-edge innovations, gaining a significant market advantage by accelerating their capabilities.
- **Innovation Leapfrogging:** This involves pursuing breakthrough developments that surpass predictable, step-by-step improvements, enabling organizations to achieve advanced outcomes much more rapidly.

Kaggle's Corporate Collaborations:

Organizations often turn to Kaggle to connect their analytical challenges with a global pool of skilled data scientists. A common obstacle many companies face is their hesitance to share proprietary datasets. Kaggle addresses this by creating a secure, competitive environment that encourages firms to release anonymized data, enabling thousands of data scientists to collaboratively tackle real-world problems.

This crowdsourced model has led to notable successes. For instance, Allstate achieved a 271% improvement in one of its actuarial models. In another case, a competition with a modest $1,000 prize generated solutions valued at over $100,000.

Some companies also use Kaggle as a recruitment tool. Facebook, for example, once organized a competition where the reward was not money, but a job interview. The event drew 422 participants, highlighting the platform’s potential for talent scouting. However, some critics, like Cathy O’Neil, argue that such initiatives might distract data scientists from considering deeper questions regarding corporate data ethics and policies.

Ethical Considerations:

Despite the benefits, several ethical concerns emerge:

- Internal employees may be at risk if external competitors develop more effective solutions than in-house teams.
- Contributors might feel exploited, especially when their efforts—driven by relatively small incentives—are used to benefit large corporations.
- While Kaggle charges companies to host competitions and offers monetary prizes, participation is optional for data scientists.

This model continues to work in favor of companies, particularly while many data scientists are eager to gain experience or exposure. However, if contributors begin to recognize the full value of their work, or demand greater transparency and ethical accountability, engagement may decline—unless the challenges align with their personal or professional values.

B) David Huffaker’s Perspective (Google)

David Huffaker from Google advocates for a mixed-methods strategy in social research, integrating both qualitative and quantitative approaches to produce deeper insights. His major contributions include:

- From Description to Prediction: Transitioning from summarizing existing data to using it for forecasting future patterns and behaviors. This involves developing predictive models grounded in historical information.
- Merging Research Approaches: Huffaker promotes the combination of methods like user interviews and ethnographic observation with large-scale data analytics. This blended methodology offers a richer, more nuanced picture of user habits and societal trends.
- Retention Modeling: Creating predictive systems that estimate whether users will remain active based on their behavior and engagement. Such models assist in designing better strategies to improve user satisfaction and loyalty.
- Ethics and Data Privacy: Huffaker places strong emphasis on responsible data usage. He highlights the need for transparency, informed consent, and giving users control over their data to build trust and uphold ethical standards.

2. Feature selection

Features:

In the realm of data science and machine learning, *features* refer to the individual measurable attributes or variables that describe an observed phenomenon. These features act as input data for algorithms and are critical in enabling models to detect patterns, make classifications, or forecast outcomes. In essence, features are the foundation upon which predictive models are built.

Feature Selection:

Feature selection is the technique of identifying the most important variables from a dataset that contribute meaningfully to a model's performance. It plays a crucial role in refining models by removing irrelevant or redundant information. This not only enhances performance but also improves the model's generalizability and interpretability.

Why Is Feature Selection Important?

- Minimizing Overfitting: Removing unimportant features helps reduce the model's complexity, allowing it to generalize better to unseen data.
- Boosting Computational Efficiency: A leaner set of features means faster training and prediction times, using fewer resources.
- Improving Interpretability: With fewer variables, models become more transparent and easier to explain to stakeholders or end users.

Approaches to Feature Selection

There are four major strategies for selecting relevant features:

- Filter Methods: These rely on statistical techniques to rank features independently of any specific machine learning algorithm. Metrics such as correlation coefficients, chi-square tests, and mutual information are commonly used.
- Wrapper Methods: These evaluate different combinations of features using a predictive model. Recursive Feature Elimination (RFE) is a well-known example that recursively removes less important variables.
- Embedded Methods: In these techniques, feature selection is part of the model training process. Algorithms like Lasso (L1 regularization) automatically shrink less important coefficients to zero during training.
- Hybrid Methods: These combine filter and wrapper techniques to leverage the strengths of both—balancing computational efficiency with predictive power.

Example of Feature Selection in Practice

Take the case of a mobile gaming app like *Chasing Dragons*. To predict whether a user will return in the following month, the model might consider features such as:

- Frequency of activity during the first month
- Duration between the first and second sessions
- Daily accumulated points
- Demographic data and device usage

Carefully selecting from these features can enhance the model's ability to make accurate predictions.

Choosing Features with Stepwise Regression

Stepwise regression is a popular method to systematically select features based on their contribution to model performance. It operates in three primary modes:

- Forward Selection: Starts with no features and adds them one at a time—choosing the feature that most improves model performance at each step—until no further improvements can be made.
- Backward Elimination: Begins with all features and removes them one by one—each time eliminating the least impactful feature—until performance no longer improves.
- Bidirectional (Stepwise) Selection**:** A hybrid approach that involves both adding and removing features. It balances the inclusion of relevant features while eliminating redundancy.

Initially, a model may use all available features, but through this iterative selection process, only the most predictive ones are retained, enhancing both efficiency and accuracy.

Criteria for Selecting the Best Model

To assess which model performs best, data scientists use several well-established evaluation criteria. These not only guide feature selection but also help compare model variants:

- R-squared (Coefficient of Determination): Indicates how much of the variation in the dependent variable is explained by the model. Higher values suggest better explanatory power.
- P-values: Especially useful in regression, p-values assess the statistical significance of each feature's coefficient. Features with low p-values are likely to have a meaningful impact on the model.
- Akaike Information Criterion (AIC): Balances model fit and complexity. It penalizes models with more parameters. Lower AIC scores imply a better trade-off between accuracy and simplicity.
- Bayesian Information Criterion (BIC): Similar to AIC but with a stricter penalty for model complexity, especially as the sample size increases. Lower BIC values suggest a more optimal model.
- Entropy (Information Gain): Commonly used in classification tasks (e.g., decision trees), entropy measures disorder in the dataset. Features that reduce entropy the most are preferred, as they contribute to clearer distinctions between classes.
- Adjusted R-squared (enhancement**):** An improved version of R-squared that accounts for the number of features in the model. It helps prevent overfitting by penalizing the addition of non-informative predictors.
- Cross-Validation Scores (enhancement): Evaluating model performance on different subsets of the data helps ensure the selected features generalize well and are not tailored to a specific sample.

By carefully selecting features and using robust evaluation metrics, data scientists can build efficient, interpretable, and high-performing models suited to a variety of tasks and domains.

3. User retention

Predictive models help determine which user behaviors lead to higher retention. For example, excessive ad exposure in the first five minutes might reduce retention. A/B testing is used to validate such insights.

User Retention Models: Developing models that predict user retention based on various factors, such as user activity and engagement patterns. These models help in strategizing to improve user retention and satisfaction.

User Retention: Interpretability Versus Predictive Power

The balance between interpretability and predictive power is important in user retention using decision trees. The key points are discussed below:

1. Interpretability vs. Predictive Power: Decision trees can predict user behavior effectively, but interpreting them for actionable insights can be challenging. For example, a straightforward insight like "more user engagement leads to higher retention" may not offer actionable strategies.

2. Insightful Interpretation: A more insightful interpretation could reveal specific actions that affect user behavior, such as avoiding showing ads in the initial minutes of app usage. This kind of insight can guide strategies to enhance user retention.

3. Causal vs. Correlation Understanding: There's a distinction between user behavior (e.g., frequency of app usage) and platform actions (e.g., displaying ads). Understanding causation versus correlation is crucial; for instance, high user engagement correlates with higher retention but doesn't necessarily cause it.

4. Feature Selection and Testing: Initial modeling and feature selection help prioritize A/B testing efforts. It's essential to test hypotheses derived from the model to validate actionable insights and refine strategies.

5. Practical Implementation: Focus on features that are actionable (e.g., optimizing ad display timings) rather than those influenced by confounding factors like user interest. This approach ensures that strategies are grounded in actionable insights derived from the data.

4. Wrappers

5. Entropy

Entropy, originally a concept from information theory, is used to measure the level of uncertainty or randomness in a system. In machine learning—particularly in algorithms like decision trees—entropy is a key metric for determining how mixed or impure a dataset is.

For a binary random variable X that can take on two outcomes (e.g., 0 and 1), entropy H(X) is calculated using the formula:

$$H(X) = -p(X = 1) \cdot \log_2 p(X = 1) - p(X = 0) \cdot \log_2 p(X = 0)$$

Where:

- p(X=1) is the probability that XXX equals 1
- p(X=0)p(X=0)p(X=0) is the probability that XXX equals 0

Key Characteristics of Entropy

- Zero Entropy (No Uncertainty): When the outcome is completely predictable—meaning one class has a probability of 1—the entropy is 0. For example, if every instance belongs to the same class, there's no uncertainty to measure.
- Maximum Entropy (Complete Uncertainty**):** When all outcomes are equally probable (e.g., p(X=1)=0.5p(X=1) = 0.5p(X=1)=0.5 and p(X=0)=0.5p(X=0) = 0.5p(X=0)=0.5), the entropy reaches its peak value of 1 bit. This indicates the highest possible level of unpredictability.

The given below figure.1 shows the sample entropy.

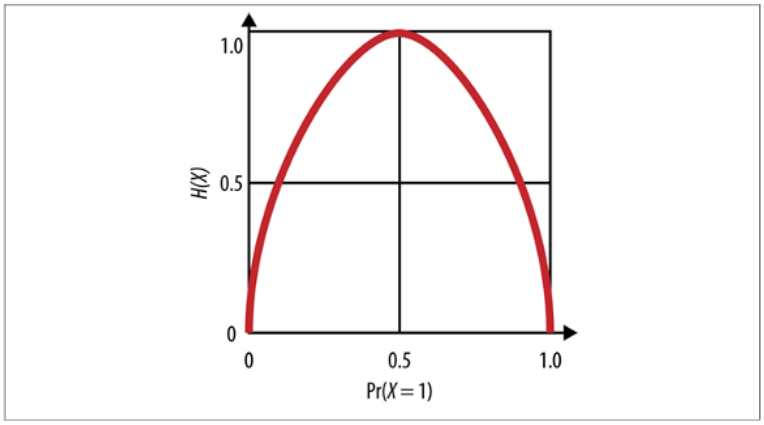


Figure: Entropy

Intuitive Examples

Entropy can be thought of as a way to gauge how "mixed" a dataset is:

- High Entropy Example: If the likelihood of a newborn being male or female is about equal, the uncertainty is high—meaning the entropy is also high.
- Low Entropy Example: In an arid desert where rain is extremely rare, the probability of rainfall is very low. This high predictability results in low entropy.

6. Decision tree algorithm

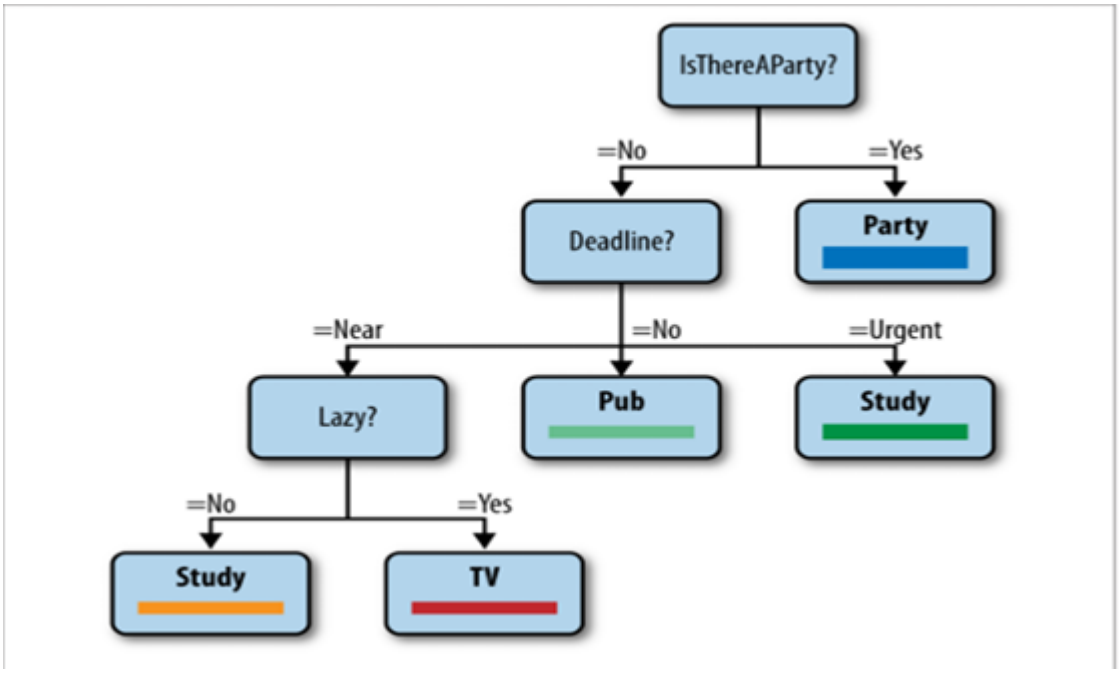
Decision trees are intuitive models that simplify complex decision-making by breaking it into a sequence of logical conditions or questions. Widely used in both practical scenarios and machine learning tasks, they serve as powerful tools for classification and regression problems.

In predictive modeling, decision trees classify instances (e.g., users) into specific categories (e.g., return or no return) based on multiple attributes such as age, activity levels, or demographics. The tree structure visually maps out these decisions from root to leaf, providing clear paths from input features to final predictions.

Core Components of a Decision Tree

- Root Node – The initial split based on the most informative feature.
- Internal Nodes – Represent decisions based on attributes.
- Branches – Indicate outcomes of those decisions.
- Leaf Nodes – Final outcomes or predicted classes.

Example: A college student deciding how to spend time might have a decision tree that starts with “Is there a party?” and follows with checks like deadlines or urgency, ending with outcomes like “Study,” “Party,” or “Watch TV.”



How Decision Trees Are Built?

- A general procedure for constructing a decision tree:
- Start with the full dataset.
- Calculate entropy for the target variable.
- Measure information gain for each feature.
- Select the feature with the highest information gain for splitting.
- Split the dataset accordingly.
- Repeat recursively for resulting subsets.
- Stop when all instances belong to the same class or no more features remain.
- Optionally, prune the tree to remove branches that don't improve accuracy, reducing overfitting.

Handling Continuous Variables

When dealing with continuous data, the goal is to convert numerical ranges into decision boundaries the tree can use. This can be handled in two ways:

i) Using Built-In Libraries (e.g., scikit-learn)

Most ML libraries automatically manage continuous features by finding optimal thresholds for splitting. Users can pass raw continuous values, and the algorithm internally determines where splits should occur.

ii) Manual Implementation of Threshold Splitting

If building a tree from scratch, handling continuous data requires:

- Finding Optimal Thresholds: Try various split points and compute metrics like information gain or Gini impurity to identify the threshold with the best performance.
- Binary Partitioning: Once a threshold is chosen, split data into two groups—values below and above or equal to the threshold—effectively converting it into a binary feature.
- Information Gain Calculation: Evaluate how much uncertainty is reduced after the split. The best threshold minimizes entropy or impurity.
- Threshold Selection as a Subproblem: Treat threshold optimization as a subroutine within the overall tree-building process.
- Optional Binning: Instead of binary splits, you can convert continuous features into discrete intervals or bins. While this simplifies the tree, it may reduce the model's granularity.

Example:
If a feature records the number of dragons a user has slain, thresholds like `<10` or `≥10` can be tested. The split that yields the highest information gain would be selected.

Best Practices and Considerations

- The method used to handle continuous data significantly affects model accuracy.
- Optimal thresholds depend on the dataset and problem context.
- While continuous features add flexibility, careful partitioning is essential for effective model performance.

7. Random forests

Random Forest is a powerful extension of decision trees that enhances prediction accuracy and robustness by using a technique known as bagging (bootstrap aggregating). While it improves performance, it can reduce model interpretability compared to individual decision trees.

Key Concepts in Random Forests:

A) Bagging (Bootstrap Aggregating):

Bagging involves creating multiple data subsets through sampling with replacement. Each of these subsets is used to train a separate decision tree. This approach:

- Reduces model variance,
- Prevents overfitting,
- Increases generalization capability.

B) Hyperparameters:

Random Forests are relatively easy to configure, with a few essential parameters:

- Number of Trees (N**):** Defines how many decision trees to grow in the forest.
- Number of Features (F): Specifies how many features to randomly consider at each split within a tree.

Bootstrapping Explained:

- In bootstrapping, data is randomly sampled with replacement**,** usually selecting about 80% of the total dataset per sample.
- Although the sample size becomes an implicit third hyperparameter, it is typically kept fixed during model training.

Steps in the Random Forest Algorithm:

A) Bootstrap Sampling:
For each tree, generate a random sample of the training data (with replacement).
B) Random Feature Selection at Each Node:
At each decision point, randomly choose *F* features from the full feature set.
C) Node Splitting:
Among the selected features, determine the best split using metrics such as information gain **or** entropy**.**
D) Tree Construction:
Grow each tree to its full depth. Unlike traditional decision trees, Random Forest trees are usually not pruned**,** allowing them to capture detailed data patterns, even if some noise is included.

Tree Depth and Complexity:

While it's possible to control tree depth or apply pruning to simplify trees, Random Forests typically leave trees unpruned to allow them to learn subtle patterns. This depth helps increase predictive power by capturing complex relationships in the data.

Random Forests in Action: Code Example (R)

Below is a detailed implementation of various models, including Random Forests, using R and the diamonds dataset:

```
# Load the diamonds dataset

require(ggplot2)

data(diamonds)

head(diamonds)



# Plot histogram of diamond prices with a reference line at $12,000

ggplot(diamonds) + geom_histogram(aes(x=price)) +

geom_vline(xintercept=12000)
```

```
# Create a binary label: 'Expensive' diamonds if price >= $12,000
diamonds$Expensive <- ifelse(diamonds$price >= 12000, 1, 0)
head(diamonds)

# Remove the original 'price' column
diamonds$price <- NULL
```

Logistic Regression using glmnet:

```
require(glmnet)
x <- model.matrix(~., diamonds[, -ncol(diamonds)])
y <- as.matrix(diamonds$Expensive)
system.time(modGlmnet <- glmnet(x = x, y = y, family = "binomial"))
plot(modGlmnet, label = TRUE)
```

Decision Tree Model

```
require(rpart)
modTree <- rpart(Expensive ~ ., data = diamonds)
plot(modTree)
text(modTree)
```

Bagging (Bootstrap Aggregation):

```
require(boot)
mean(diamonds$carat)
sd(diamonds$carat)
boot.mean <- function(x, i) { mean(x[i]) }
```

```
boot(data = diamonds$carat, statistic = boot.mean, R = 120)
```

Boosting with mboost:

```
require(mboost)

system.time(modglmBoost <- glmboost(as.factor(Expensive) ~ .,

data = diamonds, family = Binomial(link = "logit")))

summary(modglmBoost)
```

Random Forest with iris Dataset:

```
require(randomForest)

system.time(modForest <- randomForest(Species ~ ., data = iris,

importance = TRUE, proximity = TRUE))
```

Pros of Random Forest:

- Higher Accuracy: Combining multiple decision trees typically leads to better predictive performance than a single tree.
- Overfitting Resistance: The averaging of multiple trees helps prevent the model from fitting noise in the data.
- Flexibility: Works well for both classification and regression problems.
- Feature Importance Estimation: Can rank features based on their contribution to the model, offering interpretability in feature selection.

Cons of Random Forest:

- Complexity: Unlike individual decision trees, the final model becomes difficult to interpret due to the ensemble nature.
- Computational Demands: Training many trees and calculating predictions across them can be time- and resource-intensive, especially on large datasets.

8. **Singular Value Decomposition (SVD)**

Concept and Mathematical Foundation

Singular Value Decomposition (SVD) is a method to decompose any m×n matrix X of rank k into three specific matrices:

$$X = USV^T$$

Where:

- U is an m × k matrix whose columns are mutually orthogonal,
- S is a diagonal matrix of size k×k,
- V is a k × n matrix, also with orthogonal columns.

Standard SVD

- In the standard version of SVD, both U and V are typically square and unitary matrices (i.e., orthonormal and invertible).
- The matrix S, although diagonal, may be rectangular depending on the dimensions of X.

Application in Data Analysis

- The matrix XXX often represents a dataset — for instance, a user-item matrix where rows correspond to users and columns to rated items.
- The rank kkk represents the number of significant underlying patterns or relationships in the data.
- A parameter ddd, which functions similarly to the kkk in kkk-nearest neighbors, is selected to control the number of latent features to retain in the reduced representation.

Key Properties of SVD

A) Orthogonality and Ranking

- The columns of both UUU and VVV are orthogonal vectors.
- These columns can be ordered based on their associated **singular values** in SSS, allowing for a ranking from most to least impactful features.

B) Low-Rank Approximation

- **Truncation for Approximation**: A low-rank approximation of XXX can be obtained by keeping only the top ddd singular values in SSS, along with the corresponding columns in UUU and VVV.
- **Data Compression**: This reduces the size and complexity of the data while retaining the most relevant information.

C) Dimensionality Reduction via Latent Variables

- By choosing a value ddd smaller than kkk, the data is transformed into a reduced latent space, simplifying its structure without significant information loss.

D) Latent Feature Interpretation

- The decomposed matrices UUU and VVV uncover **latent patterns or features** in the dataset.
- For example, a leading latent feature might correspond to demographic factors such as gender or preferences, distinguishing between different user groups.

Challenges and Limitations

- Handling Incomplete Data**:** SVD alone does not account for missing values within the dataset and often requires preprocessing or imputation.
- Computational Demand: Performing SVD on large-scale datasets is resource-intensive, both in terms of memory and processing time, making it less practical for massive real-time applications.

SVD Algorithm: Step-by-Step Guide

Step 1: Start with Matrix X

Given an $m \times n$ matrix X, our goal is to decompose it into three matrices U, S, and V^T.

Step 2: Compute XX^T

Calculate the product of X and its transpose:
XX^T

Step 3: Find Eigenvalues of XX^T

Solve the characteristic equation to find the eigenvalues of XX^T:
$\det(XX^T - \lambda I) = 0$

Step 4: Calculate Singular Values

The singular values σ_i are the square roots of the eigenvalues found in Step 3.

Step 5: Compute Right Singular Vectors (V)

Calculate the eigenvectors of X^TX corresponding to the eigenvalues found in Step 3. These eigenvectors form the columns of V.

Step 6: Compute Left Singular Vectors (U)

Using the singular values and the right singular vectors, compute the left singular vectors using:
$u_i = \frac{1}{\sigma_i} X v_i$
where u_i are the columns of U, and v_i are the columns of V.

Step 7: Form the Diagonal Matrix (S)

Create the diagonal matrix S with the singular values σ_i on the diagonal and zeros elsewhere.

Step 8: Construct the SVD

Combine the matrices U, S, and V^T to form the SVD of X:

$$X = USV^T$$

.

9. **Principal Component Analysis (PCA)**

Principal Component Analysis (PCA) is a dimensionality reduction technique often used to predict user preferences by approximating a given data matrix X with the product of two lower-dimensional matrices U and V^T, such that:

$$X \approx U \cdot V^T$$

The goal is to minimize the difference between the original data X and the approximation $U \cdot V^T$, typically measured by the sum of squared errors.

Prediction Interpretation:

- Each row u_i in matrix U represents a specific user's latent preferences.
- Each row v_j in matrix V corresponds to a particular item's latent attributes.
- The predicted value of user i's preference for item j is given by the dot product $u_i \cdot v_j$.
- The optimization aims to align this predicted value as closely as possible with the actual entry $x_{i,j}$ in the data matrix.

Formulating the Optimization Problem:

The objective is to find matrices U and V that minimize the overall squared difference between the known values in matrix X and the predicted values derived from $U \cdot V^T$. This process closely mirrors the approach used in linear regression to minimize the mean squared error.

Latent Dimensions and Feature Space:

- One critical hyperparameter in PCA is the number of latent dimensions d, which defines the number of features retained in the decomposition.
- The matrices U and V are of size m × d and n × d, respectively.
- In practical applications, a value of d around 100 is often used to balance between preserving meaningful structure and ensuring computational efficiency.

Uniqueness and the Impact of Missing Data:

- Although PCA aims to find a low-dimensional representation of the data, the decomposition is not unique**,** especially in datasets with missing entries.

- Despite this non-uniqueness, obtaining *any* valid decomposition that closely reconstructs the observed entries is typically sufficient for downstream applications.

Uncorrelated Components:

One of the key advantages of PCA is that the latent features (or principal components) are uncorrelated. This decorrelation reduces redundancy, making the resulting model more compact and efficient.

Core Properties of PCA:

A) Matrix Approximation:
If matrices U and V are identified such that $U \cdot V^T \approx X$ with minimum error, they serve as effective representations of the original data.
B) Transformation Invariance:
If G is any invertible d × d matrix, then replacing U with UG and V with VG^{-T} will still approximate X accurately, i.e., $UG \cdot (VG^{-T})^T = U \cdot V^T$.
C) Orthogonality through Minimization:
By seeking to minimize the surface area of the ddd-dimensional parallelepiped formed by the columns of U, while preserving its volume, the columns become mutually orthogonal—ensuring that latent components are uncorrelated.
D) Orthogonality in **V**:
When U's columns are orthogonal, the structure of PCA guarantees that the rows of V are orthogonal as well, reinforcing the uncorrelated nature of the latent features.

Additional Theoretical Insights:

- Determinant and Scaling Considerations:
When transforming the factor matrices U and V using scaling matrices with non-unit determinants, the optimal scaling factor (in terms of minimizing the sum of squared entries across U and V) is the geometric mean of the scales.
- Relationship to SVD:
PCA shares foundational concepts with Singular Value Decomposition (SVD)**.** While they are not identical, PCA is often implemented using SVD algorithms due to their mathematical similarities and efficiency.

Here's how PCA works in simple terms:

A) Data Simplification: o Imagine you have a dataset with many features (e.g., height, weight, age, income). PCA helps to reduce this to a smaller set of new features that still capture the essential information.
B) Finding Principal Components: o PCA identifies new features called "principal components." These are combinations of the original features that explain the most variance (differences) in the data. The first principal component captures the most variance, the second captures the next most, and so on.
C) Transformation: o The original data is transformed into a new set of uncorrelated features (principal components). These new features are easier to work with because they are not redundant and capture the most important patterns in the data.

D) Dimensionality Reduction: By selecting a few principal components, you can reduce the number of features while retaining most of the important information. This makes the data simpler and faster to process without losing significant insights.

Why Use PCA?

- Simplification: Makes complex data easier to understand and visualize.
- Efficiency: Reduces computational resources needed for analysis and modeling.
- Noise Reduction: Helps to remove less important information and focus on the most critical aspects of the data.
- Uncorrelated Features: Ensures that the new features are not redundant, which can improve the performance of machine learning models.

Benefits:

- Reduces noise
- Helps visualize high-dimensional data
- Speeds up training

Module 3

Plotting and Visualization

Developing clear and insightful visualizations—often referred to as plots—is a key element of data analysis. Visuals serve different purposes: they can be used in the early stages of exploration to detect anomalies or assess the need for data transformation, or they may act as the final product, such as interactive dashboards for web-based applications.

In Python, a wide range of libraries is available to support both static and interactive visualizations. Among these, matplotlib is one of the most widely used libraries, especially for creating high-quality 2D visualizations suitable for publication. It was originally developed by John Hunter in 2002 to replicate MATLAB-style plotting functionality in Python.

Over the years, the matplotlib project has worked closely with the IPython/Jupyter ecosystem to make it easier to create interactive plots directly within Jupyter notebooks. The library is compatible with various graphical user interface (GUI) backends across different operating systems and supports exporting visuals in numerous vector and image formats, such as PDF, SVG, PNG, JPG, GIF, BMP, and others.

Most of the figures presented in this textbook were generated using matplotlib. Furthermore, a number of higher-level visualization libraries have been built on top of matplotlib. One such popular library is seaborn, which is discussed later in the chapter.

To work with matplotlib interactively in a Jupyter notebook, you can activate inline plotting by executing:

```
%matplotlib notebook
```

This enables a dynamic interface for generating and interacting with plots directly within the notebook environment.

1. A Brief matplotlib API Primer

1.1. Figures and Subplots

To start using matplotlib, a popular Python library for visualizations, you typically import its pyplot module with the following line of code:

import matplotlib.pyplot as plt

If you're working within a Jupyter Notebook, enabling inline or interactive plotting is helpful. You can do this using:

%matplotlib notebook

This sets up the notebook to render plots directly within the interface. Once configured, you can begin creating basic plots such as line graphs. For instance:

```
In [12]: import numpy as np

In [13]: data = np.arange(10)

In [14]: data
Out[14]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [15]: plt.plot(data)
```

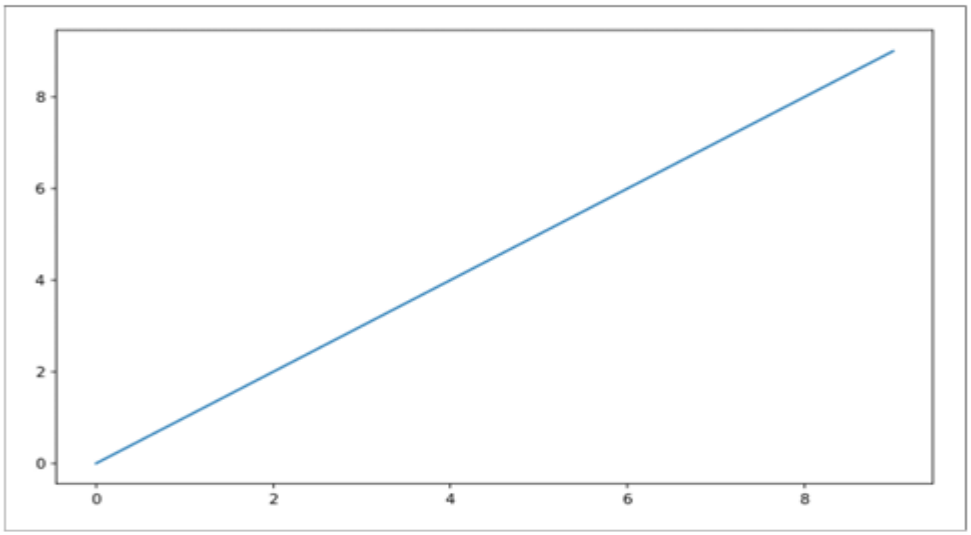


Figure 1: Simple line plot

This line command creates a simple line chart using default figure and subplot settings.

A) Figure and Subplot Structure:

In matplotlib, every plot lives inside a Figure object. You can manually create a figure using:

```
fig = plt.figure()
```

At this point, the figure exists but will be empty unless you add a subplot. Subplots are added using:

```
ax1 = fig.add_subplot(2, 2, 1)
```

This means you want to divide the figure into a 2x2 grid and create (or select) the first ed similarly.

```
In [18]: ax2 = fig.add_subplot(2, 2, 2)

In [19]: ax3 = fig.add_subplot(2, 2, 3)
```

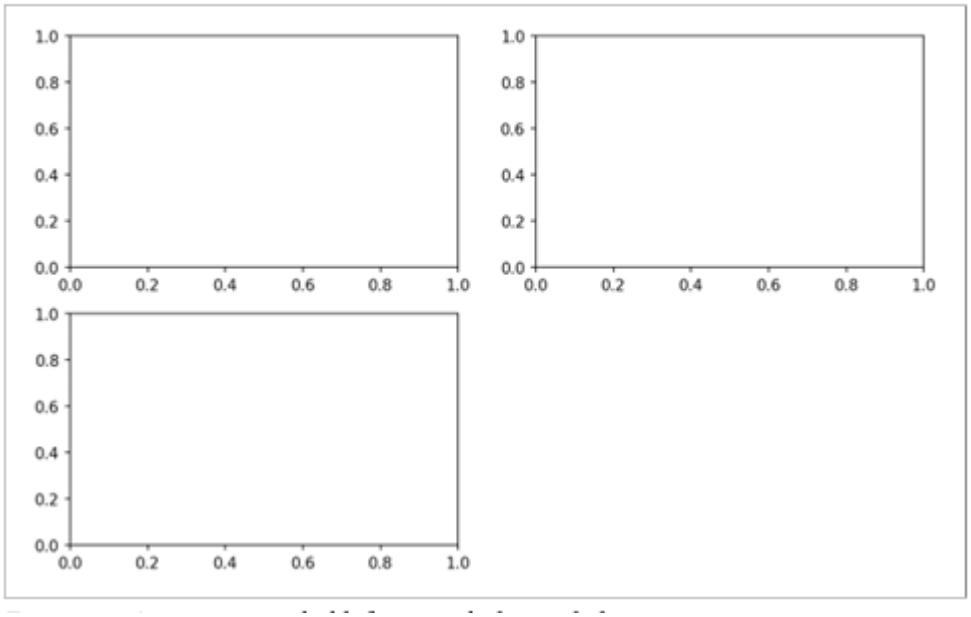


Figure 2: An empty matplotlib figure with three subplots

Here, run the given commands in the same cell:

fig = plt.figure()

ax1 = fig.add_subplot(2, 2, 1)

ax2 = fig.add_subplot(2, 2, 2)

ax3 = fig.add_subplot(2, 2, 3)

plt.plot([1.5, 3.5, -2, 1.6])

While executing, mat plotlib draws on the last figure and subplot used. So, add the following command, to create subplot like Figure 3.

plt.plot(np.random.randn(50).cumsum(), 'k--')

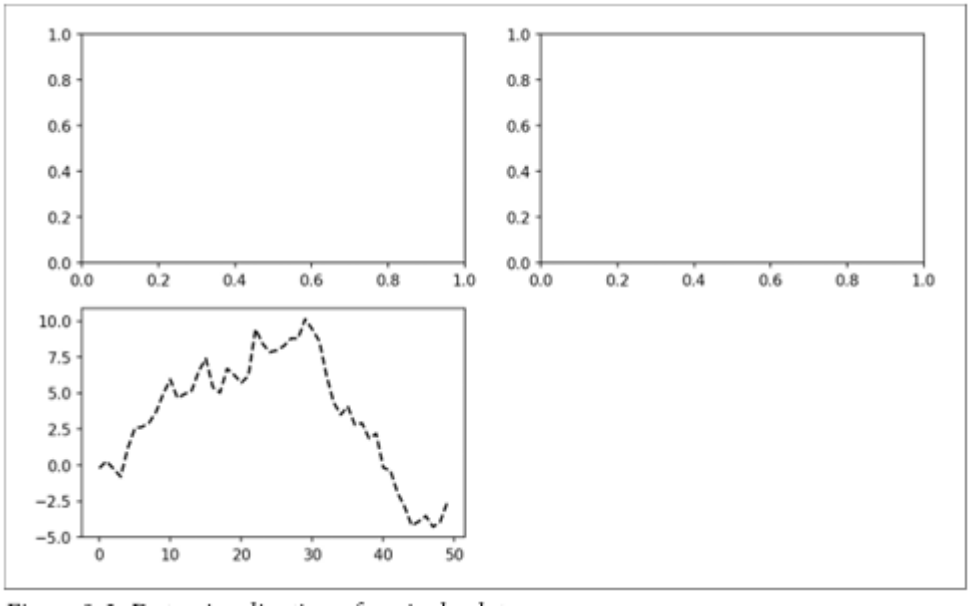


Figure 3: Data visualization after single plot

To style the plot, matplotlib supports format strings like `'k--'`, which plots a **black dashed line**:

```
In [21]: _ = ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)

In [22]: ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

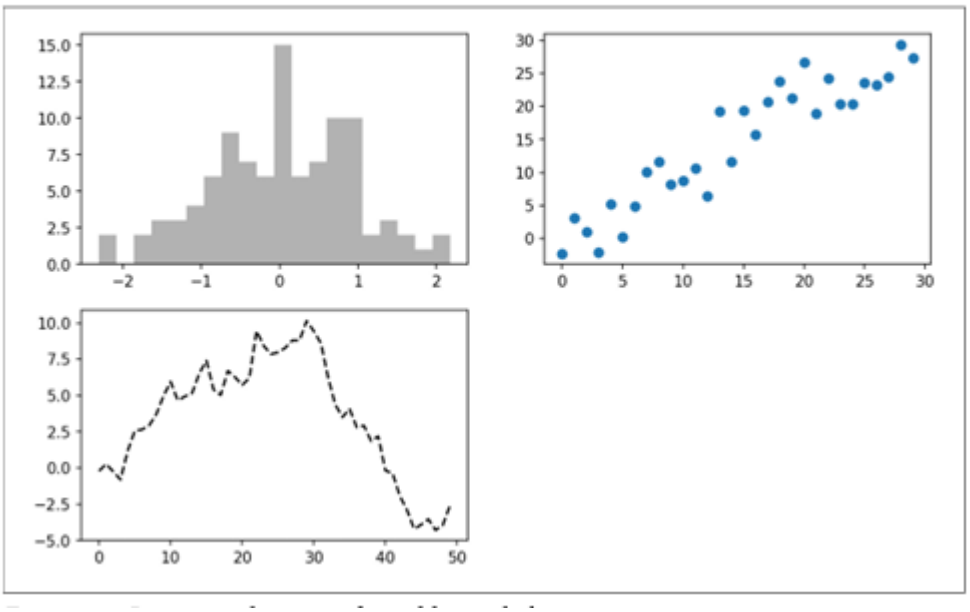


Figure 4: Data visualization after additional plots

The axes array can be easily indexed like a 2-D array, for example, axes[0, 1]. Like this, we can indicate the subplots with same x- or y-axis using sharex and sharey, respectively. This is

particularly useful when you compare data on the same scale; otherwise, matplotlib autoscales plot limits independently. Refer Table 1 for structure.

Table 1. pyplot.subplots options

Argument	Description
`nrows`	Number of rows of subplots
`ncols`	Number of columns of subplots
`sharex`	All subplots should use the same x-axis ticks (adjusting the `xlim` will affect all subplots)
`sharey`	All subplots should use the same y-axis ticks (adjusting the `ylim` will affect all subplots)
`subplot_kw`	Dict of keywords passed to `add_subplot` call used to create each subplot
`**fig_kw`	Additional keywords to `subplots` are used when creating the figure, such as `plt.subplots(2, 2, figsize=(8, 6))`

B) Adjusting the spacing around subplots:

By default, matplotlib adds spacing around and between subplots. This is scaled relative to the figure size and adjusts dynamically if you resize the plot.

To fine-tune this spacing, use the `subplots_adjust()` method. It can be called from the figure object or as a global function:

```
plt.subplots_adjust(left=..., right=..., top=..., bottom=..., wspace=..., hspace=...)
```

- `wspace` controls horizontal spacing (width between subplots)
- `hspace` controls vertical spacing (height between subplots)

Example:
To remove all space between subplots:

```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(np.random.randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=0, hspace=0)
```

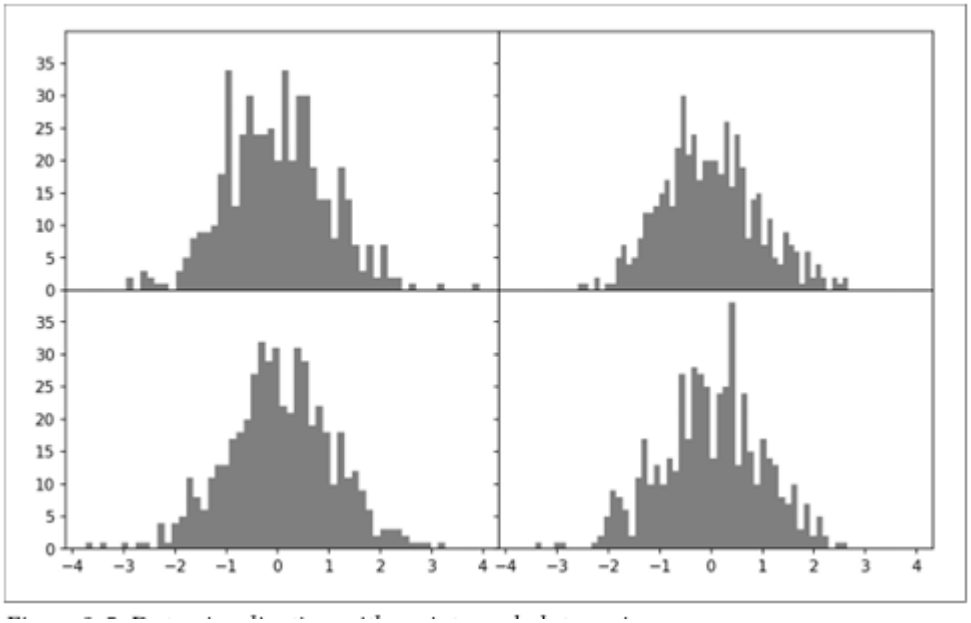


Figure 5: Data visualization with no inter-subplot spacing

1.2. Colors, Markers, and Line Styles

Matplotlib's core plot function accepts arrays of x and y coordinates and alternatively a string abbreviation indicating color and line style. For instance, to plot x versus y with green dashes, execute:

```
ax.plot(x, y, 'g--')
```

While combining color and line style in a single string offers convenience, it's not always ideal—especially when generating plots programmatically. In such cases, it's often better to specify styles separately rather than assembling them into one string. The same plot can be created more clearly using explicit style parameters.

```
ax.plot(x, y, linestyle='--', color='g')
```

Line charts can include markers to emphasize individual data points. Because matplotlib connects points with continuous lines, it's not always obvious where the actual data values are positioned. To make this clearer, markers can be added directly within the style string—this string typically follows

the order of color, marker type, and then line style (refer to Figure 6 for illustration).

```
In [30]: from numpy.random import randn

In [31]: plt.plot(randn(30).cumsum(), 'ko--')
```

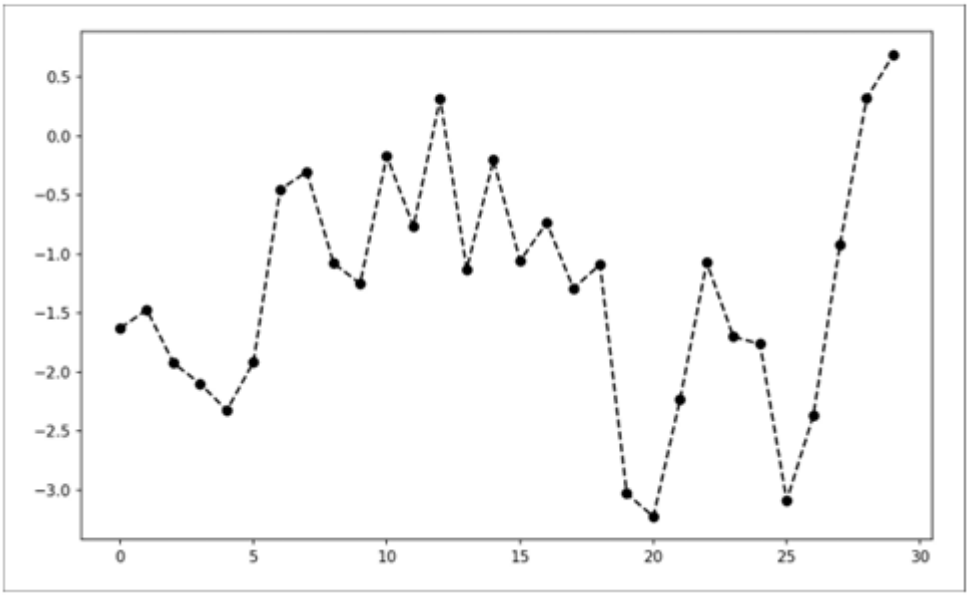


Figure 6: Line plot with markers

For line plots, subsequent points are linearly interpolated by default, which can be changed with the drawstyle option (Figure 7):

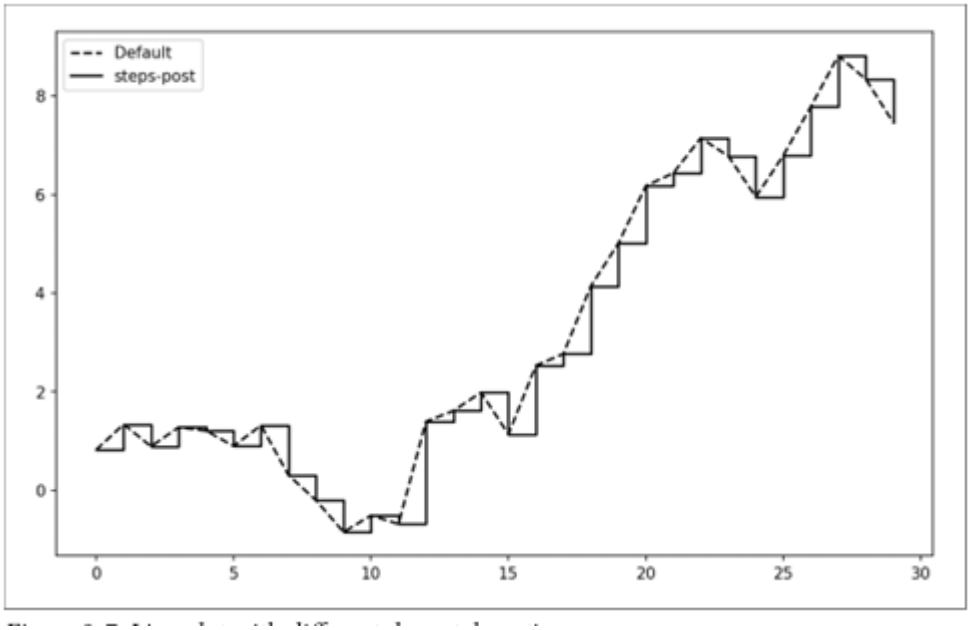


Figure 7: Line plot with different drawstyle options

1.3. Ticks, Labels, and Legends

When it comes to customizing plots in matplotlib, there are generally two approaches you can take: the **procedural method** using the `pyplot` module (e.g., `matplotlib.pyplot`), or the more **object-oriented method** using the core matplotlib API.

The `pyplot` interface is designed for quick, interactive plotting and includes functions like `xlim`, `xticks`, and `xticklabels`, which control aspects such as axis limits, tick positions, and tick labels.

These functions can be used in two ways:

- Without arguments: Returns the current setting (e.g., `plt.xlim()` gives the current x-axis limits).
- With arguments: Updates the setting (e.g., `plt.xlim([0, 10])` sets the x-axis range from 0 to 10).

By default, these functions apply to the most recently active `AxesSubplot` object. Internally, each of these commands has an equivalent pair of methods on the subplot object itself. For example, `xlim` corresponds to `ax.get_xlim()` and `ax.set_xlim()`.

While both methods are valid, using the object-oriented approach with subplot instances (e.g., `ax.set_xlim`) is often clearer—especially when dealing with multiple subplots. It allows for more precise control, though users can choose whichever style they find more comfortable.

A) **Setting the title, axis labels, ticks, and ticklabels**

Refer, figure 8 for customizing the axws.

```
In [37]: fig = plt.figure()

In [38]: ax = fig.add_subplot(1, 1, 1)

In [39]: ax.plot(np.random.randn(1000).cumsum())
```

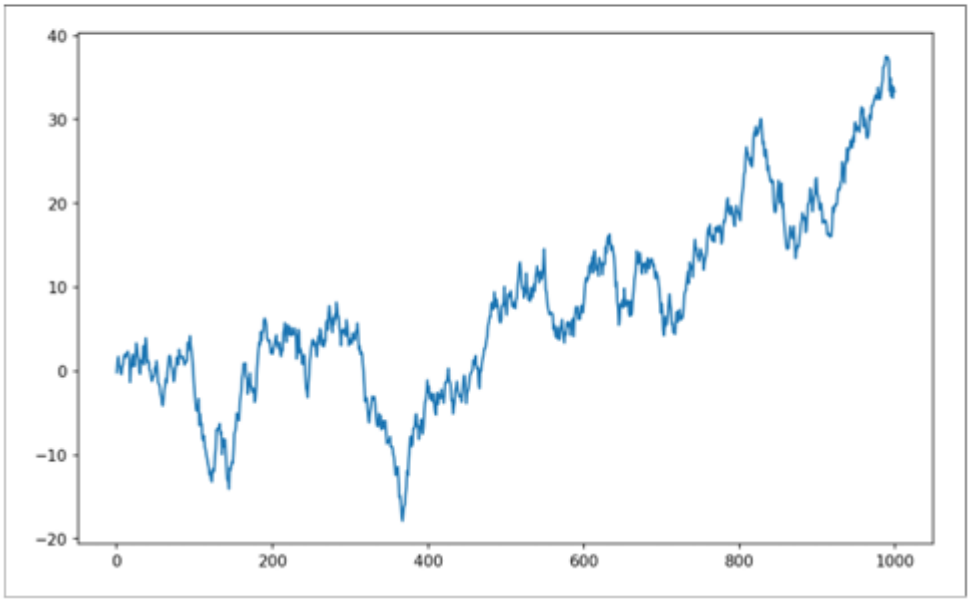


Figure 8: Simple plot for illustrating xticks (with label)

To change the x-axis ticks, use set_xticks and set_xticklabels.

Example,

In [40]: ticks = ax.set_xticks([0, 250, 500, 750, 1000])

In [41]: labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'], ….: rotation=30, fontsize='small')

The rotation option rotates the x tick labels at a 30-degree. Lastly, set_xlabel gives a name to the x-axis and set_title the subplot title (Refer figure 9)

: In [42]: ax.set_title('My first matplotlib plot')

Out[42]: In [43]: ax.set_xlabel('Stages')

In [43]: ax.set_xlabel('Stages')

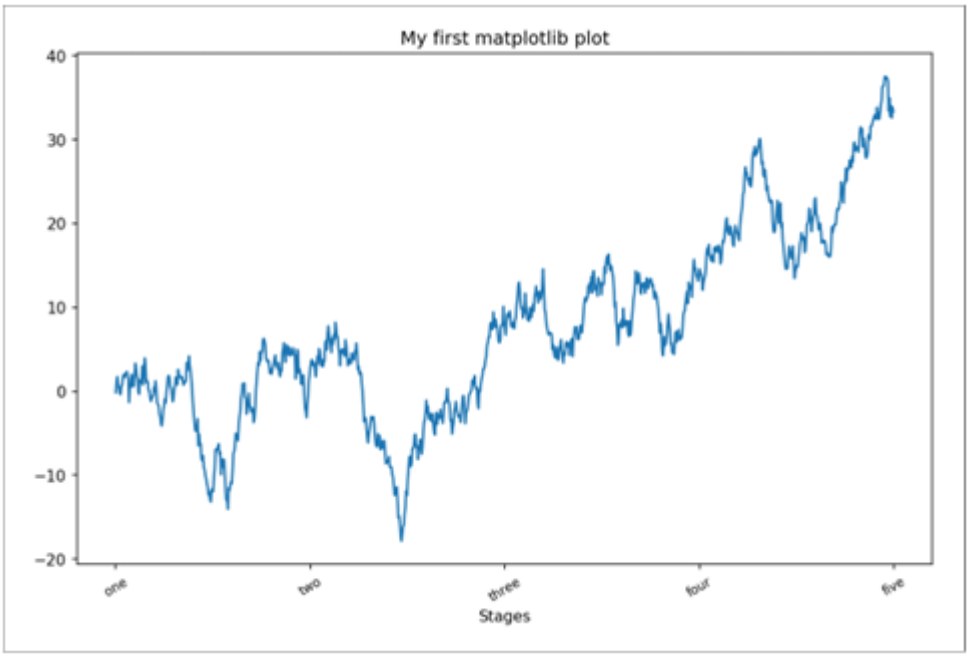


Figure 9: Simple plot for illustrating xticks

B) **Adding legends**

Legends are one of the critical elements to identify plot elements. There are a many ways, and the easiest is to pass the label argument when adding each piece of the plot:

```
In [44]: from numpy.random import randn

In [45]: fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)

In [46]: ax.plot(randn(1000).cumsum(), 'k', label='one')
Out[46]: [<matplotlib.lines.Line2D at 0x7fb624bdf860>]

In [47]: ax.plot(randn(1000).cumsum(), 'k--', label='two')
Out[47]: [<matplotlib.lines.Line2D at 0x7fb624be90f0>]

In [48]: ax.plot(randn(1000).cumsum(), 'k.', label='three')
Out[48]: [<matplotlib.lines.Line2D at 0x7fb624be9160>]
```

After that, either call ax.legend() or plt.legend() to create a legend automatically (Refer figure 10)

```
In [49]: ax.legend(loc='best')
```

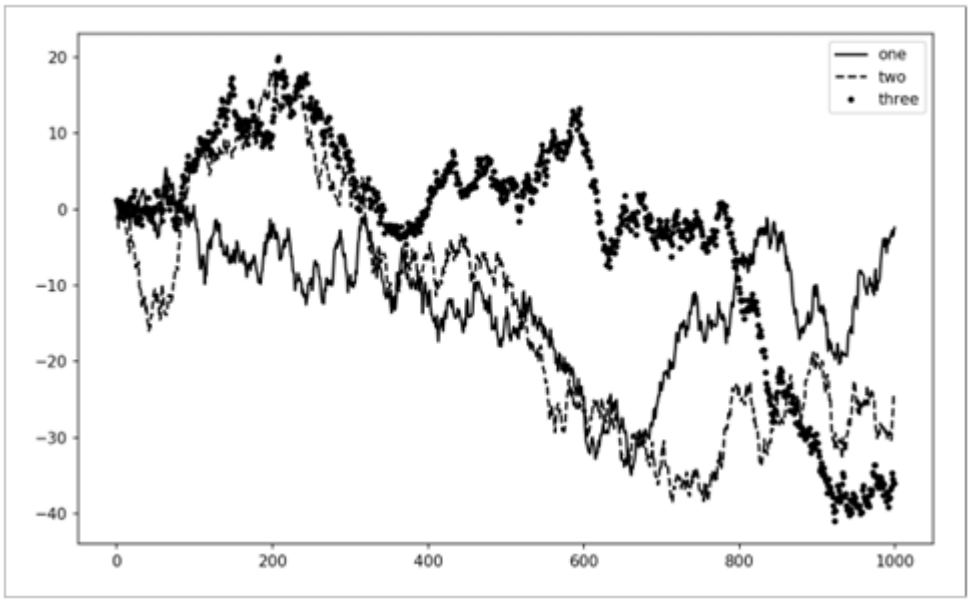


Figure 10: Simple plot with three lines and legend

1.4. Annotations and Drawing on a Subplot

To draw your plot annotations, which could consist of text, arrows, or other shapes, add annotations and text using the text, arrow, and annotate functions. 'text draws text' at specified coordinates (x, y) on the plot with optional custom styling:

```
ax.text(x, y, 'Hello world!',
        family='monospace', fontsize=10)
```

The annotations shown above add both text and arrows to the plot in a visually organized way. For instance, you could visualize the S&P 500 closing prices from 2007 onward (data available via Yahoo! Finance) and mark significant events from the 2008–2009 financial crisis directly on the graph. The entire example can be conveniently executed in a single cell within a Jupyter notebook (Refer figure 11).

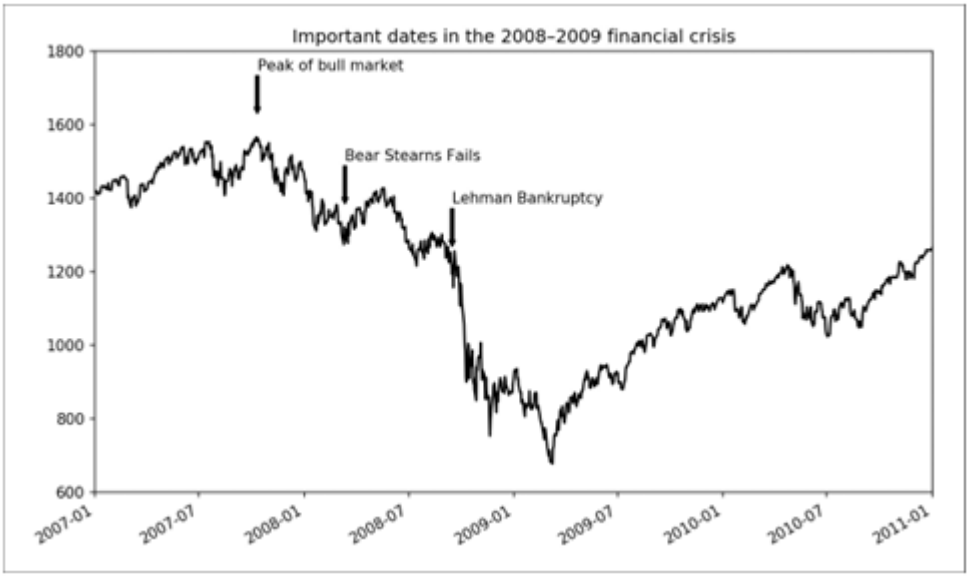


Figure 11. Important dates in the 2008–2009 financial crisis

To draw geometric shapes, matplotlib offers built-in objects for many standard forms such as Rectangle and Circle. While some are accessible through matplotlib.pyplot, the complete collection is available in the matplotlib.patches module. To include a shape in your plot, you first create the patch object (e.g., shp) and then attach it to a subplot using ax.add_patch(shp) (refer to Figure 12 for illustration).

Example:

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

rect = plt.Rectangle((0.2, 0.75), 0.4, 0.15, color='k', alpha=0.3)
circ = plt.Circle((0.7, 0.2), 0.15, color='b', alpha=0.3)
pgon = plt.Polygon([[0.15, 0.15], [0.35, 0.4], [0.2, 0.6]],
                   color='g', alpha=0.5)

ax.add_patch(rect)
ax.add_patch(circ)
ax.add_patch(pgon)
```

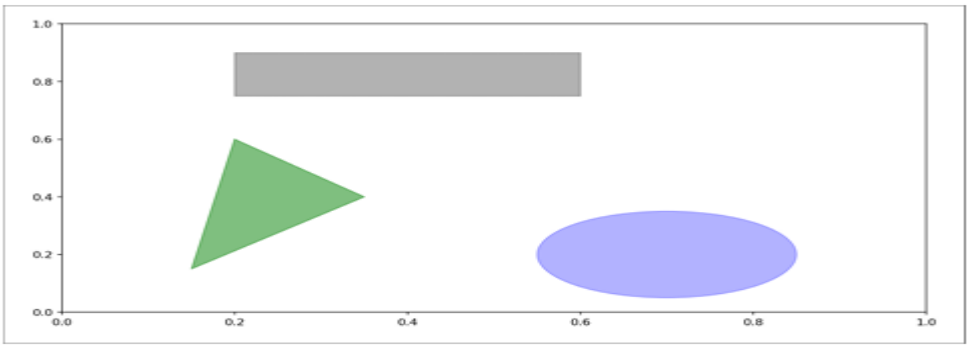


Figure 12. Data visualization composed from three different patches

1.5. Saving Plots to File

You can export the current figure to a file using the plt.savefig function, which performs the same action as calling the savefig method on a figure object. For instance, to save a plot in SVG format, simply use:

```
plt.savefig('figpath.svg')
```

The file type is automatically determined by the file extension you provide. So if you save it with a .pdf extension instead, a PDF file will be generated.

When preparing figures for publication, two commonly adjusted options are:

- dpi: Controls the resolution in dots per inch.
- bbox_inches='tight': Removes excess whitespace around the figure.

To save a high-resolution PNG file with minimal padding, you could write:

```
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```

In addition to saving directly to a file, savefig can also output the plot to a file-like object such as a BytesIO stream. This is useful when you need to process the image in memory:

```
from io import BytesIO
buffer = BytesIO()
plt.savefig(buffer)
plot_data = buffer.getvalue()
```

For more customization options, refer to the full list in Table 2.

Table 2. Figure.savefig options

Argument	Description
`fname`	String containing a filepath or a Python file-like object. The figure format is inferred from the file extension (e.g., `.pdf` for PDF or `.png` for PNG)
`dpi`	The figure resolution in dots per inch; defaults to 100 out of the box but can be configured
`facecolor`, `edgecolor`	The color of the figure background outside of the subplots; `'w'` (white), by default
`format`	The explicit file format to use (`'png'`, `'pdf'`, `'svg'`, `'ps'`, `'eps'`, ...)
`bbox_inches`	The portion of the figure to save; if `'tight'` is passed, will attempt to trim the empty space around the figure

1.6. matplotlib Configuration

By default, **matplotlib** is set up with color palettes and layout options suited for creating figures intended for publication. However, nearly every default setting—such as figure dimensions, spacing between subplots, font size, color schemes, grid appearance, and more—can be fully customized.

One way to adjust these settings programmatically is by using the `rc()` function. For example, to globally set the default figure size to **10 by 10**, you would write:

```
plt.rc('figure', figsize=(10, 10))
```

The first argument passed to `rc()` specifies the component to customize (like `'figure'`, `'axes'`, `'xtick'`, `'ytick'`, `'grid'`, or `'legend'`). Additional settings can be supplied as keyword arguments.

Another convenient approach is to define settings in a dictionary and apply them with unpacking. For example:

```
font_options = {'family' : 'monospace',
                'weight' : 'bold',
                'size'   : 'small'}
plt.rc('font', **font_options)
```

For broader and more persistent customization, matplotlib provides a config file named **`matplotlibrc`**, located in the `matplotlib/mpl-data` directory. You can also create your own version of this file (named `.matplotlibrc`) and save it in your home directory. When matplotlib starts, it will automatically apply these custom settings.

As you'll learn later, the seaborn library builds on matplotlib and includes several pre-defined themes and styles, all managed through this same configuration system.

2. Plotting with pandas and seaborn

matplotlib operates at a relatively low level, requiring users to build a plot by combining its fundamental elements—such as the data visualization type (e.g., line, bar, scatter, box, contour), along with the title, axis labels, legends, and other annotations.

In contrast, pandas makes visualization easier, especially when working with DataFrames or Series that include multiple data columns and labeled rows and columns. It offers built-in plotting functions that streamline the process.

Additionally, the seaborn library—developed by Michael Waskom—is a high-level interface built on top of matplotlib. It simplifies the creation of commonly used statistical plots, making it easier to produce attractive and informative graphics.

2.1. Line Plots

Both **Series** and **DataFrame** objects in **pandas** include a `plot` attribute that allows users to create basic visualizations. When you call the `plot()` method without specifying a plot type, it defaults to generating a **line plot** (as shown in Figure 13).

```
In [60]: s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))

In [61]: s.plot()
```

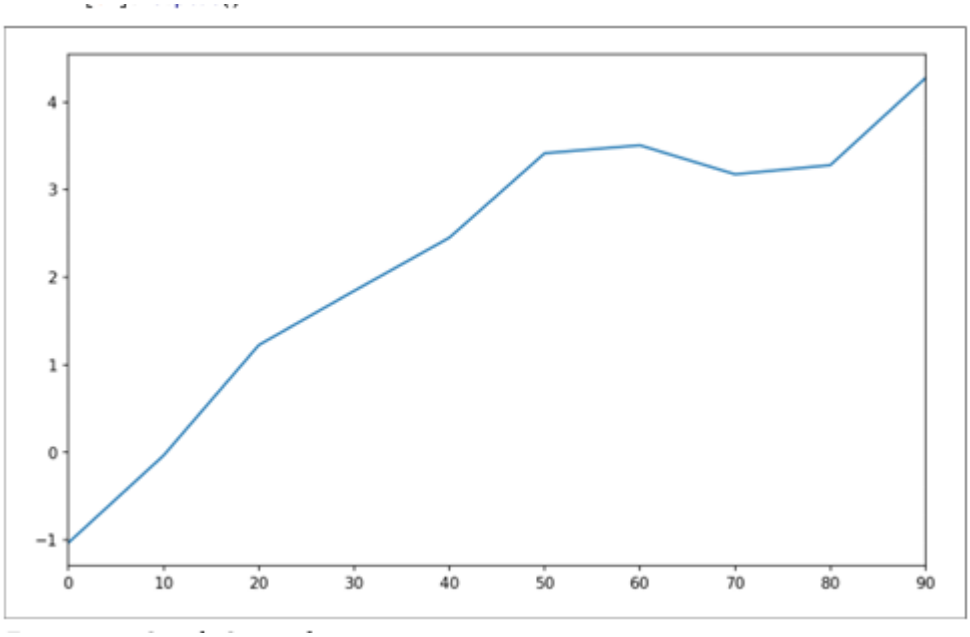


Figure 13. Simple Series plot

By default, the Series index is used as the x-axis values in the plot. If you prefer not to use the index, you can disable it by setting `use_index=False`. Axis tick marks and range limits can be customized using parameters such as `xticks` and `xlim` for the x-axis, and `yticks` and `ylim` for the y-axis. Table 3 provides a complete list of available plotting options. While some of these options are explained further in this section, you are encouraged to explore the rest independently.

Additionally, most plotting functions in pandas accept an optional `ax` argument, which allows you to pass a matplotlib subplot object**.** This is especially helpful when organizing multiple plots within a grid layout.

When using `plot()` on a DataFrame**,** each column is plotted as a separate line within the same subplot, and a legend is automatically generated (as demonstrated in Figure 14).

```
In [62]: df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),
   ....:                   columns=['A', 'B', 'C', 'D'],
   ....:                   index=np.arange(0, 100, 10))

In [63]: df.plot()
```

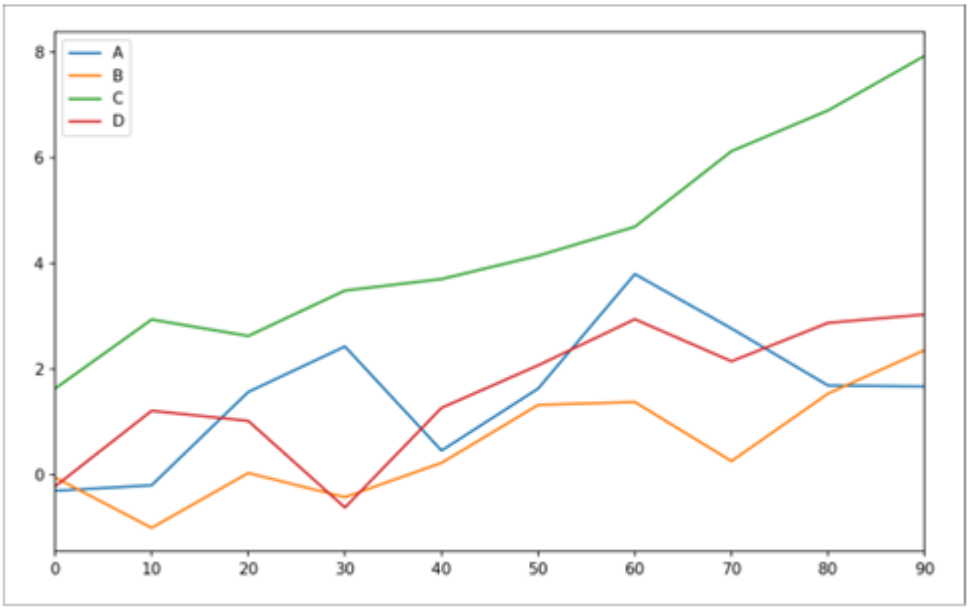


Figure 14. Simple DataFrame plot

Table 3. Series.plot method arguments

Argument	Description
`label`	Label for plot legend
`ax`	matplotlib subplot object to plot on; if nothing passed, uses active matplotlib subplot
`style`	Style string, like `'ko--'`, to be passed to matplotlib
`alpha`	The plot fill opacity (from 0 to 1)
`kind`	Can be `'area'`, `'bar'`, `'barh'`, `'density'`, `'hist'`, `'kde'`, `'line'`, `'pie'`
`logy`	Use logarithmic scaling on the y-axis
`use_index`	Use the object index for tick labels
`rot`	Rotation of tick labels (0 through 360)
`xticks`	Values to use for x-axis ticks
`yticks`	Values to use for y-axis ticks
`xlim`	x-axis limits (e.g., `[0, 10]`)
`ylim`	y-axis limits
`grid`	Display axis grid (on by default)

DataFrame allows some flexibility to handle the columns; for example, whether to plot all columns on the same subplot or to create separate subplots (refer to table 4 for more details).

Table 4. DataFrame-specific plot arguments

Argument	Description
subplots	Plot each DataFrame column in a separate subplot
sharex	If subplots=True, share the same x-axis, linking ticks and limits
sharey	If subplots=True, share the same y-axis
figsize	Size of figure to create as tuple
title	Plot title as string
legend	Add a subplot legend (True by default)
sort_columns	Plot columns in alphabetical order; by default uses existing column order

2.2. Bar Plots

To create vertical and horizontal bar plots, use plot.bar() and plot.barh() respec tively. Here, the Series or DataFrame index will use as the x (bar) or y (barh) ticks (Figure 15):

```
In [64]: fig, axes = plt.subplots(2, 1)

In [65]: data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))

In [66]: data.plot.bar(ax=axes[0], color='k', alpha=0.7)
Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb62493d470>

In [67]: data.plot.barh(ax=axes[1], color='k', alpha=0.7)
```

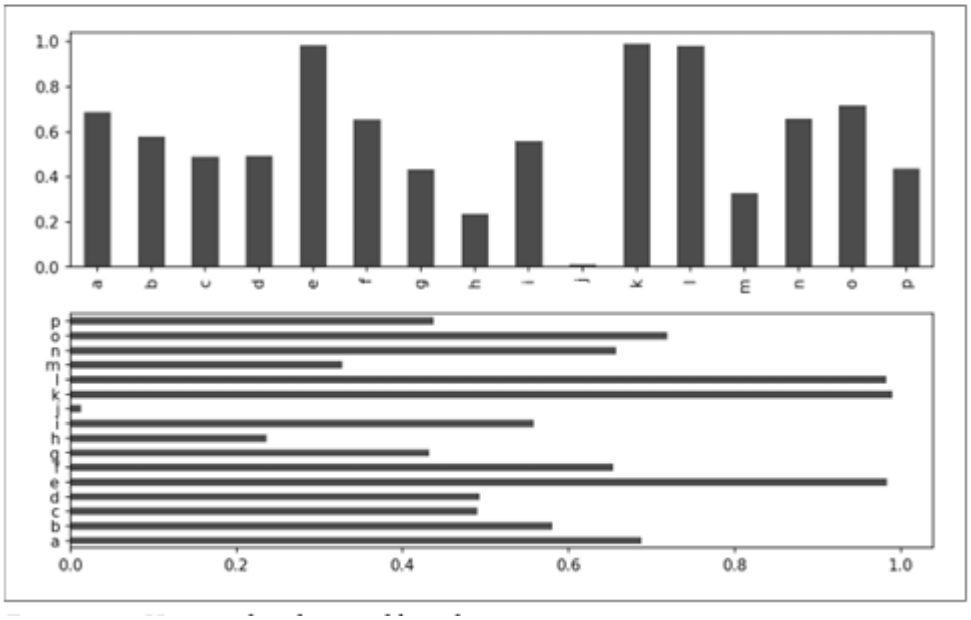


Figure 15. Horizonal and vertical bar plot

The options such as color='k' and alpha=0.7, gives black color to the plots and use partial transparency for filling.

When creating a bar plot from a DataFrame, the values from each row are displayed as grouped bars, positioned side by side to represent each data point (Figure 16).

```
In [69]: df = pd.DataFrame(np.random.rand(6, 4),
   ....:                   index=['one', 'two', 'three', 'four', 'five', 'six'],
   ....:                   columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))

In [70]: df
Out[70]:
Genus         A         B         C         D
one    0.370670  0.602792  0.229159  0.486744
two    0.420082  0.571653  0.049024  0.880592
three  0.814568  0.277160  0.880316  0.431326
four   0.374020  0.899420  0.460304  0.100843
five   0.433270  0.125107  0.494675  0.961825
six    0.601648  0.478576  0.205690  0.560547

In [71]: df.plot.bar()
```

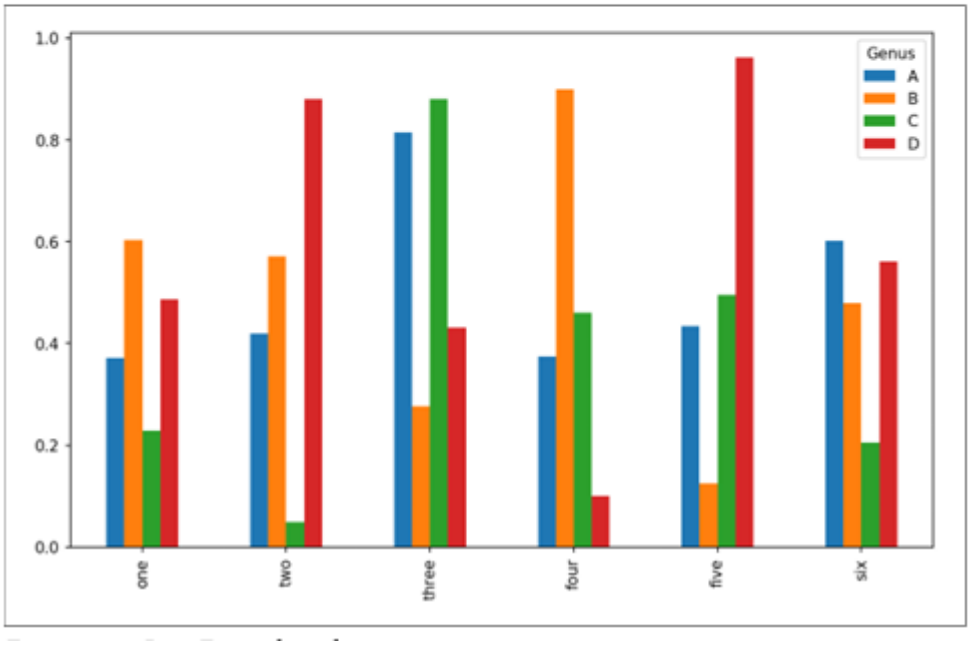


Figure 16. DataFrame bar plot

Here, “Genus” is the title of the legend

To generate stacked bar plots from a DataFrame use stacked=True (Figure 17)

```
In [73]: df.plot.barh(stacked=True, alpha=0.5)
```

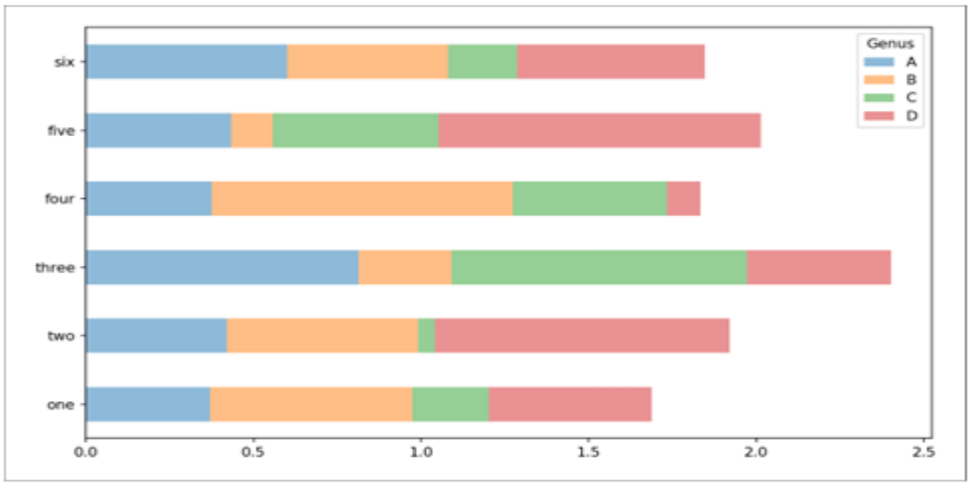


Figure 17. DataFrame stacked bar plot

To make a stacked bar plot showing the percentage of data points (for example, each party size on each day)

```
In [75]: tips = pd.read_csv('examples/tips.csv')

In [76]: party_counts = pd.crosstab(tips['day'], tips['size'])

In [77]: party_counts
Out[77]:
size  1   2   3   4  5  6
day
Fri   1  16   1   1  0  0
Sat   2  53  18  13  1  0
Sun   0  39  15  18  3  1
Thur  1  48   4   5  1  3


# Not many 1- and 6-person parties
In [78]: party_counts = party_counts.loc[:, 2:5]
```

Then, normalize (Figure: 18)

```
# Normalize to sum to 1
In [79]: party_pcts = party_counts.div(party_counts.sum(1), axis=0)

In [80]: party_pcts
Out[80]:
size         2         3         4         5
day
Fri   0.888889  0.055556  0.055556  0.000000
Sat   0.623529  0.211765  0.152941  0.011765
Sun   0.520000  0.200000  0.240000  0.040000
Thur  0.827586  0.068966  0.086207  0.017241

In [81]: party_pcts.plot.bar()
```

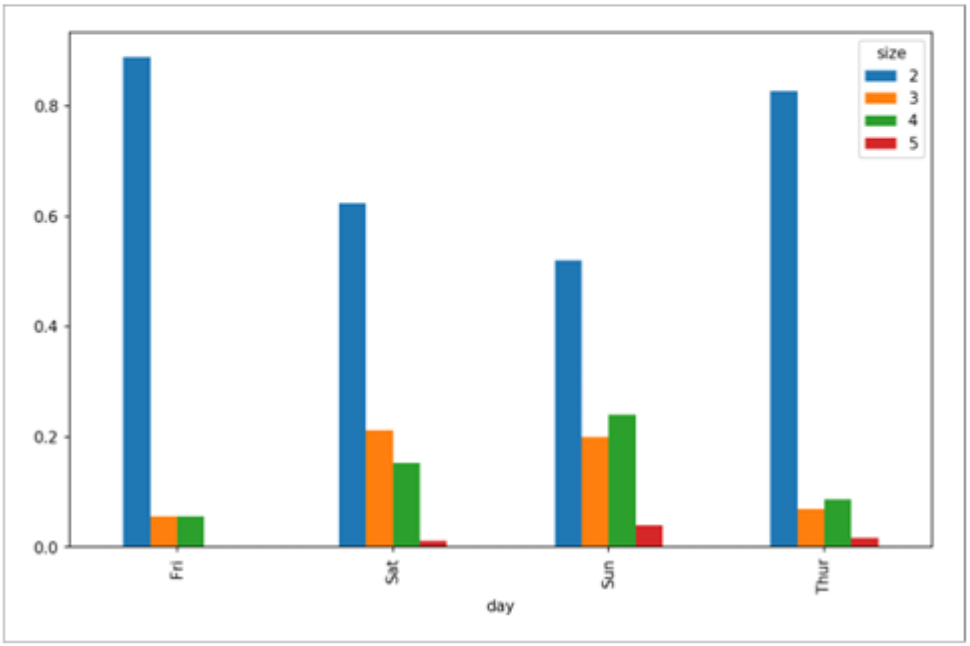


Figure 18. Fraction of parties by size on each day

For aggregation or summarization before making a plot, using the seaborn package. For example, tipping percentage by day (Figure: 19)

```
In [83]: import seaborn as sns

In [84]: tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])

In [85]: tips.head()
Out[85]:
   total_bill   tip smoker  day    time  size   tip_pct
0       16.99  1.01     No  Sun  Dinner     2  0.063204
1       10.34  1.66     No  Sun  Dinner     3  0.191244
2       21.01  3.50     No  Sun  Dinner     3  0.199886
3       23.68  3.31     No  Sun  Dinner     2  0.162494
4       24.59  3.61     No  Sun  Dinner     4  0.172069

In [86]: sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
```

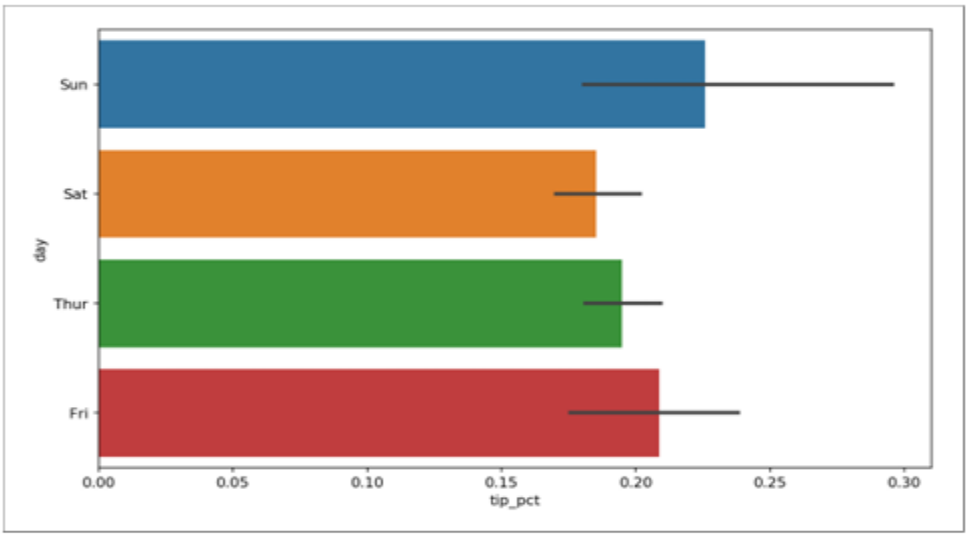


Figure 19. Tipping percentage by day with error bars

To split by an additional categorical value, seaborn.barplot use the hue option (Figure 20).

```
In [88]: sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')
```

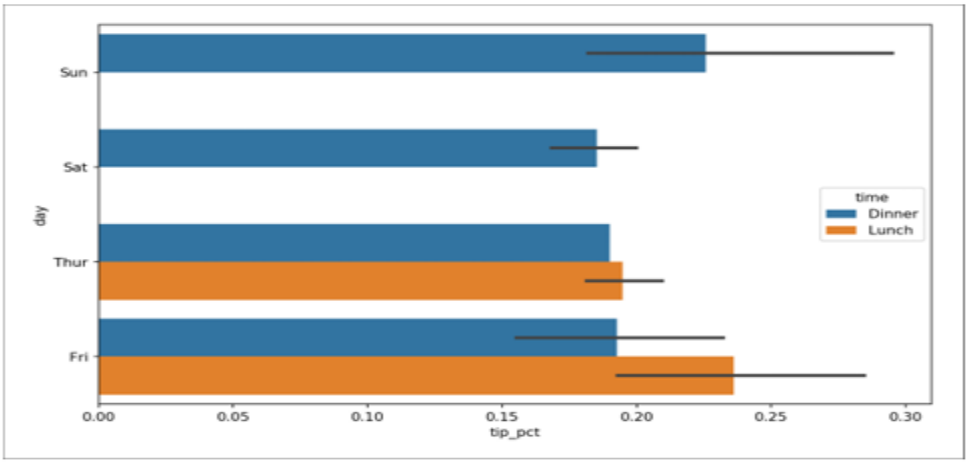


Figure 20. Tipping percentage by day and time

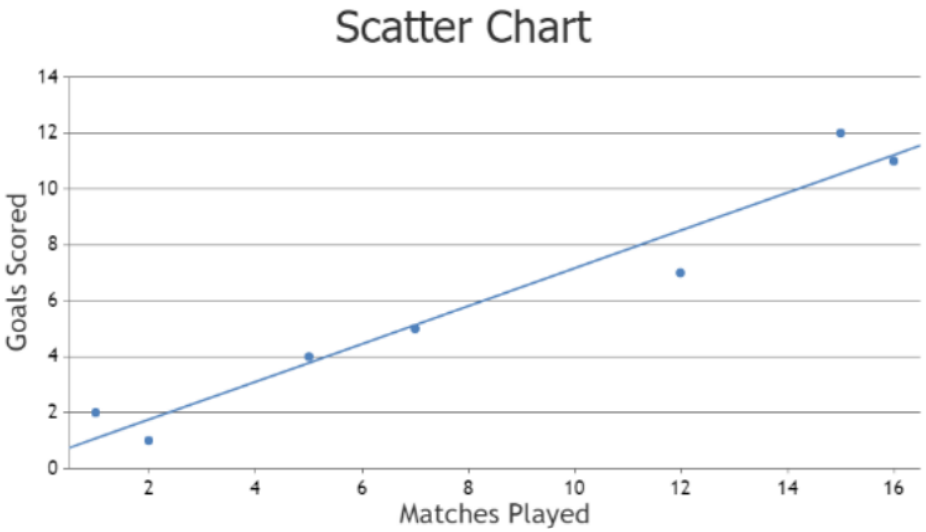
Scatter plot :

One of the most significant data visualization methods is scatter plot and it is regarded to be one of the Seven Basic Tools of Quality. The relationship of two variables is plotted in the chart called a scatter plot on two dimensional graph commonly referred to as Cartesian Plane on mathematical basis.

It is normally employed to draw the association between one independent variable and one dependent variable with an independent variable plotted on x- axis and a dependent variable plotted on y-axis so that you can determine the influence that the independent variable has on the dependent variable. These are called Scatter Plot Graph, or Scatter Diagram.

Scatter Plot has other names and some of them include; scatter chart, scattergram, scatter plot, and XY graph. Scatter plot is the graphical representation of a data pair where each element of the pair is assigned an axis that normally is the independent axis, called the x-axis and the other axis, the dependent axis, called the y-axis.

Such type of distribution facilitates the easy determination of the type of relationship, the data pair in plotted format is assuming. Thus the Scatter Plot comes in handy when we have to determine the relationship between any two sets of data or in cases where we are suspecting that there might be some relationship between the two variables and the relationship might be the gist of a problem being encountered somewhere.



We need to observe the steps below in order to create a scatter plot.

Identify the independent and dependent variables (step 1)

Step 2: On x-axis put the independent variable

Step 3: Leave the y-axis blank and plots the dependent variable

Step 4 The significant connection between the provided variables should be pulled out.

Forms of Scatter Plot

The Scatter Plot on the basis of correlation of two variables will fall under the types as following.

Scatter Plot Of Positive Correlation

Scatter Plot Of Negative Correlation

Scatter Plot of Null Correlation

Scatter Plot of Positive Correlation

In such kind of scatter-plot values on Y axis rises as one moves along from left to right. Technically speaking, when two variables are directly proportional to each other then there will be positive correlation as portrayed in the scatter plot. The familiarity is further discussed in the form of Perfect Positive, High Positive and Low Positive.

Scatter Plot Of Negative Correlation

In such scatter-plot, the y measure declines as we go left to right. Valuation of one variable against the other is reducing, in layman terms. There are further forms of positive correlation that are referred to as Perfect Negative, High Negative and Low Negative.

Scatter Plot of Null Correlation

Values in such a category of scatter-plot are distributed everywhere in the graph. Generally such a type of graph indicates that there exists no correlation between the two variables drawn in the Scatter Plot.

What is an Analysis of Scatter Plot?

The use of scatter plot analysis entails analysis of how the points on the graph are distributed and arriving to the general picture in order to get insights into the association between the variables. Scatter Plot helps to visualise correlated variables of two variables; however, in life, we are not that lucky to obtain only a correlation between variables. Such are the cases in real life cases, and more than two variables can be the part of correlation.

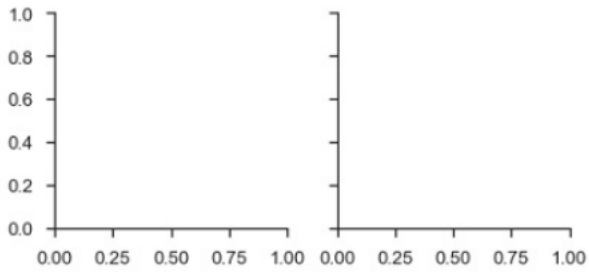
Facet grids and Categorical data:

FacetGrid object requires a dataframe and the labels of the variables which will appear in the row, column or hue dimensions of the grid.

The variables must be categorical and the data at every level of the variable is to be used as a facet along that dimension.In the given example we have simply created the facetgrid object which does not draw on them.

Forming visual data of this grid is mainly done using the FacetGrid.map() approach. Let us turn to the tip distributions on each of these subsets, a histogram can be used.

```
import pandas as pd
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('tips')
g = sb.FacetGrid(df, col = "time")
plt.show()
```



Other Python visualization tools

In data science, distribution of your data is very important. Histograms are useful in visualizing frequency distributions as are box plots (box-and-whisker). Violin plots deliver one step further the box plots in combination with the kernel density estimate (KDE), providing a more detailed outlook into distribution shape. It is also useful that KDE plots can easily be used to display smoothly calculated distributions, particularly when we want to compare several groups of data.

In categorical data, strip plot and swarm plot give the information about data distribution inside categories. Strip plots provide not-overlapping points by making it rougher (jitter) whilst swarm plot arrange the data in order to avoid overlapping at all, these are the best when one has small-to-medium provided data to work with. The stacked area chart is applied when comparing over time the composition of several categories. When data is hierarchical, then the proportion is visualized in tree maps and sunburst charts, and it is useful in showing categories and subcategories.

When dealing with geographical data, one idea is that of color-coded regions (such as countries or states) these processes work out well via the use of choropleth maps. Point maps can be used to see a specific geographical location on a latitude-longitude scale in maps and heatmaps are maps that show the intensity of data, like the number of people in some locations. These involve libraries such as folium, geopandas, plotly to make the tasks easier.

When working with analyzed relationships or interactions (e.g. social networks or web traffic), network graphs can be very helpful. You can render interconnectivity between entities (node) and connections (edge) using networkx or pyvis in an interactive way. On the same note, correlation tables that are sometimes represented in the form of heat maps can make you rapidly understand the quality of the association between numerical variables. Hierarchical clustering utilizes dendrograms, which are used to indicate the formation of data clusters depending on similarity.

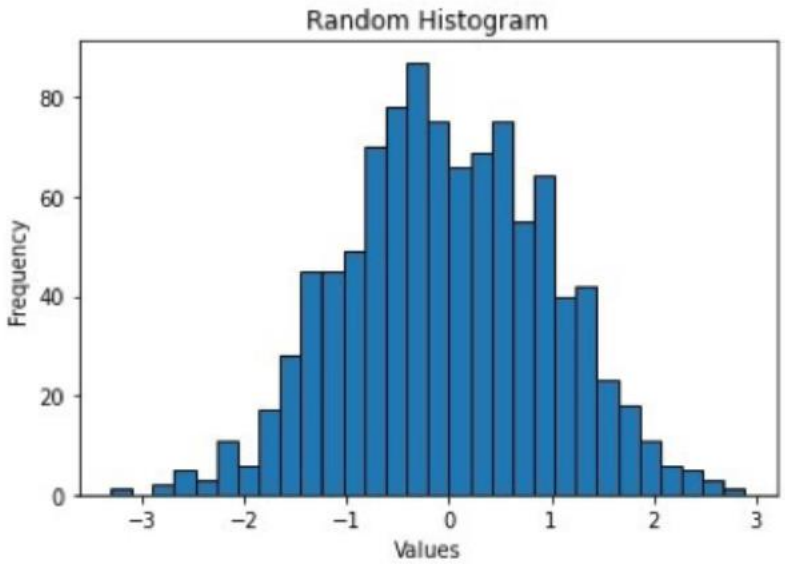
Module 4

Statistical Thinking

Distributions – Representing and plotting histograms

Histogram data is a fundamental tool in visualization, which provides a simple yet powerful way to understand the distribution of data. Whether you are new to data analysis or looking to sharpen your skills, histogram is an important tool for summarizing and imagining the data points.

A type of histogram is a type of graphical representation that refers to the distribution of numerical values. It consists of a set of vertical bars, where each time represents a range of values, and the height of the bar indicates the frequency or count of data points falling within that range.



Histograms are usually used to imagine the size of the data set in data and data analysis and identify the pattern, such as the presence of outlair or obliqueness. They are also useful to compare the distribution of various data sets or to identify trends over time.

The picture above shows how 1000 random data points from a general distribution and standard deviation of 1 are plotted into a histogram with 30 coaches and black edges.

We can create a histogram using Matplotlib by following a series of steps. Following the import details of libraries, the code generates a set of 1000 random data points from a general distribution with 0 and standard deviations, using 0 and standard deviations.

Python has a powerful tool for making plt.hist () function histogram. By providing data, number of compartments, bar color, and edge color as input, this function produces a histogram plot.

To enhance visualization, Xlabel (), ylabel (), and title () functions are used to add the labeled to X and Y axes, as well as a title for the plot.

Finally, the show () function is employed to display histograms on the screen, which allows for detailed analysis and interpretation.

```
import matplotlib.pyplot as plt
import numpy as np

# generate data
data = np.random.normal(0, 1, 1000)

# create histogram
plt.hist(data, bins=30, color='blue',
edgecolor='black')

# set labels and title
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Histogram of Data')

# display plot
plt.show()
```

Outliers

Outlier are extreme values that differ from most other data points in a dataset. They can have a major impact on your statistical analysis and slant the results of any hypothesis tests. It is important to carefully identify the potential outliers in your dataset and to treat them properly for accurate results. There are four ways to identify outliers: Sorting method Data visualization system Statistical test (jade score) Contradictory limit system The outlier has values at the extreme ends of a dataset.

Descriptive figures often involve using some numbers abbreviated to distribution. An important aspect of a distribution is where its center is located. Central trend measures are first discussed. Another aspect of a distribution is how it spreads. In other words, how different the number in distribution is from each other. The second section describes measures of variability. Distribution can vary in size. Some distribution are symmetrical while others have long tails in just one direction. The third section describes measures of distribution size. Concern of the last two volumes (1) how changes affect the measures summarizing distribution and (2) variance sum law, a significant relationship that includes a measure of variability.

🔹 1. Central Tendency (Where the data is centered)

Measure	Description	Function (Python - Pandas)
Mean	Average value	`df['col'].mean()`
Median	Middle value when data is sorted	`df['col'].median()`
Mode	Most frequent value	`df['col'].mode()`

🔹 2. Dispersion (How spread out the data is)

Measure	Description	Function (Pandas)
Range	Difference between max and min	`df['col'].max() - df['col'].min()`
Variance	Average of squared deviations from the mean	`df['col'].var()`
Standard Deviation	Spread around the mean (square root of variance)	`df['col'].std()`
IQR (Interquartile Range)	Middle 50% of data (Q3 - Q1)	`Q3 - Q1`

🔹 3. Shape of Distribution

- **Skewness**: Measures asymmetry of data
  - Positive skew → tail on right
  - Negative skew → tail on left
  - `df['col'].skew()`
- **Kurtosis**: Measures "tailedness" of distribution
  - High kurtosis → heavy tails (outliers)
  - `df['col'].kurt()`

Variance

The variance is a statistical measurement that is used to determine the spread of numbers in a data set in relation to the average value or mean. The standard deviation class will give us variance. Using the variance we can evaluate how or squeezed the distribution.

Statistics can have two types of versions, namely, sample variance and population variance. The symbol of variance is given by σ2. The variance is widely used in hypothesis tests, examining the good of the fit, and monte carlo sampling. To check how individual data points vary in relation to using variance. In this article, we will take a look at the qualities of definition, example, formula, application and variance. Variation is a solution to spread. A remedy for spread is a volume that is used to check the variability of data about the average value. Data can be of two types - grouped and ungreed. When data is expressed as square interval, it is known as grouped data. On the other hand, if the data has individual data points, it is called non -unguided data. Both sample and population variance can be determined for data.

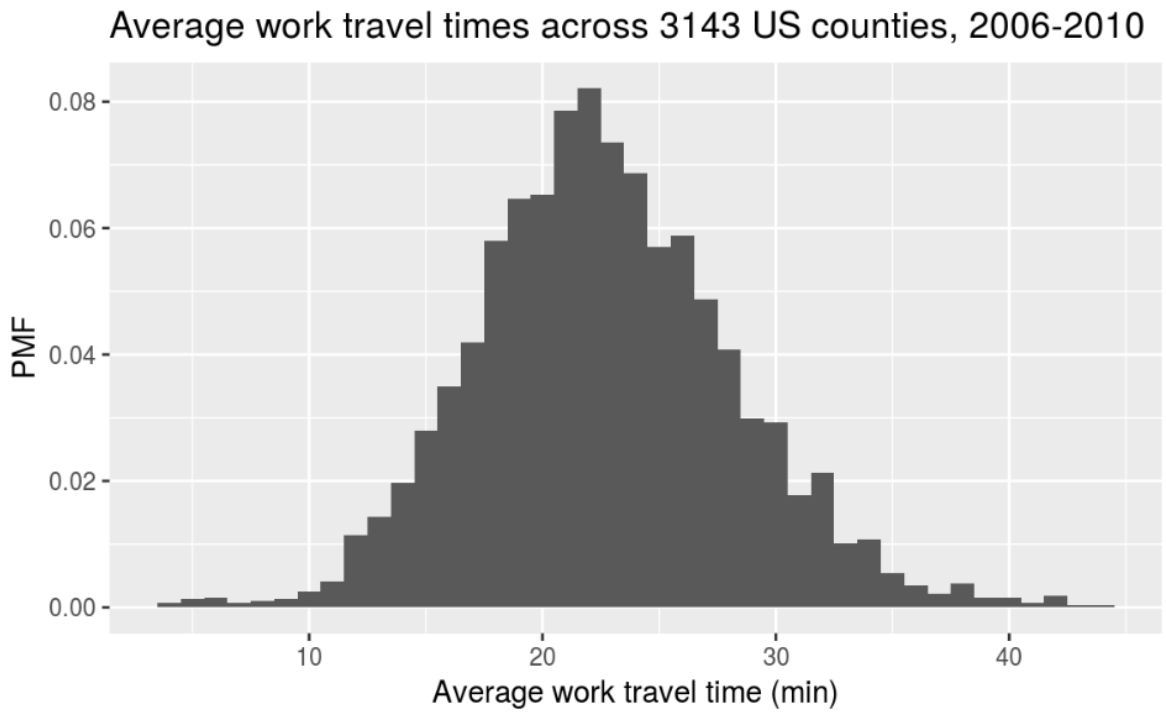
	Population	Sample
Ungrouped	$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N}$	$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N - 1}$
Grouped	$\sigma^2 = \sum_{i=1}^{N} \frac{f(M_i - \bar{X})^2}{N}$	$\sigma^2 = \sum_{i=1}^{N} \frac{f(M_i - \bar{X})^2}{N - 1}$

Probability mass function – Plotting PMFs

The probability mass function (PMF) represents a distribution by sorting the data into the compartment (a lot of frequency is like histogram) and then combines a possibility with each bin in delivery. A possibility is a frequency expressed as a fraction of sample size n. Therefore we can directly convert a frequency histogram by dividing the count in each bin in PMF by dividing sample size n. This process is called normalization.

The syntax for plotting PMF using ggplot2 is almost the same what you will use to create a frequency histogram. An amendment is that you need to include y = .. density .. inside AES (). As a simple example, let's do the complete distribution of average work time in advance and plot it as PMF:\

```
county_complete %>%
  ggplot(mapping = aes(x = mean_work_travel, y = ..density..)) +
  geom_histogram(binwidth = 1) +
  labs(title = "Average work travel times across 3143 US counties, 2006-2010",
       x = "Average work travel time (min)",
       y = "PMF")
```



Other visualizations

1. Line Graph

The line graph is the favorite law in the enumeration of graphs as well as various different number of business applications due to the fact they present a trend in a total examined manner.

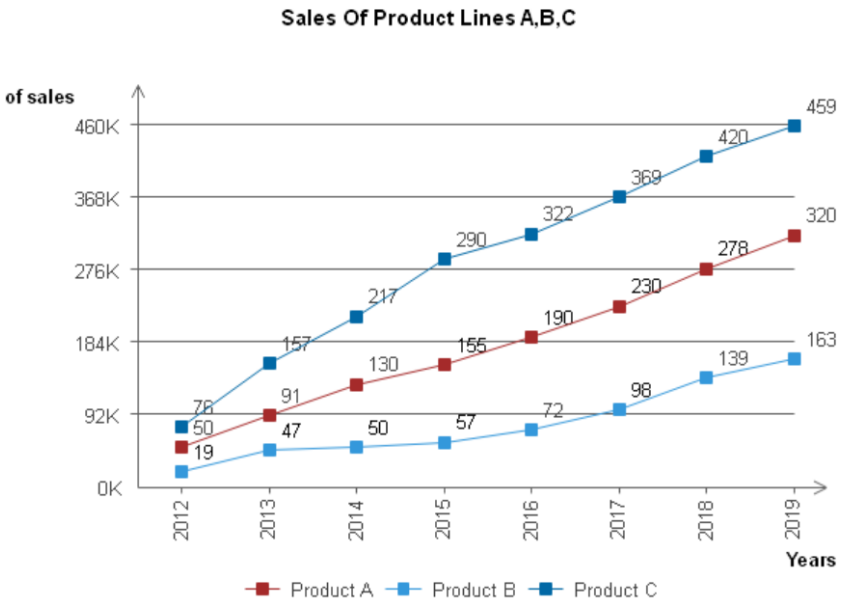
What is line graph?

A line chart (also called a line graph) is a graph that is used to represent the values of something against a given time span.

In the case of your firm, your sales department might place a graph of how the number of sales you have on hand varies with time. The values are shown in forms of data points, connected with straight lines.

When is line graphs use?

You should use it when you need to show trends.You want to present trends of various categories at the same time of a period and to present comparison.



Bar Chart

Bar charts are quite popular types of data displays since you can read them without much effort and draw useful information and they are excellent to compare multiple various types of data.

What is a bar chart?

Bar chart (or bar graph) is a type of a chart used to display data in a form of bars of various height.

The bars are of two types i.e. vertical or horizontal. You do not have to use a particular type.

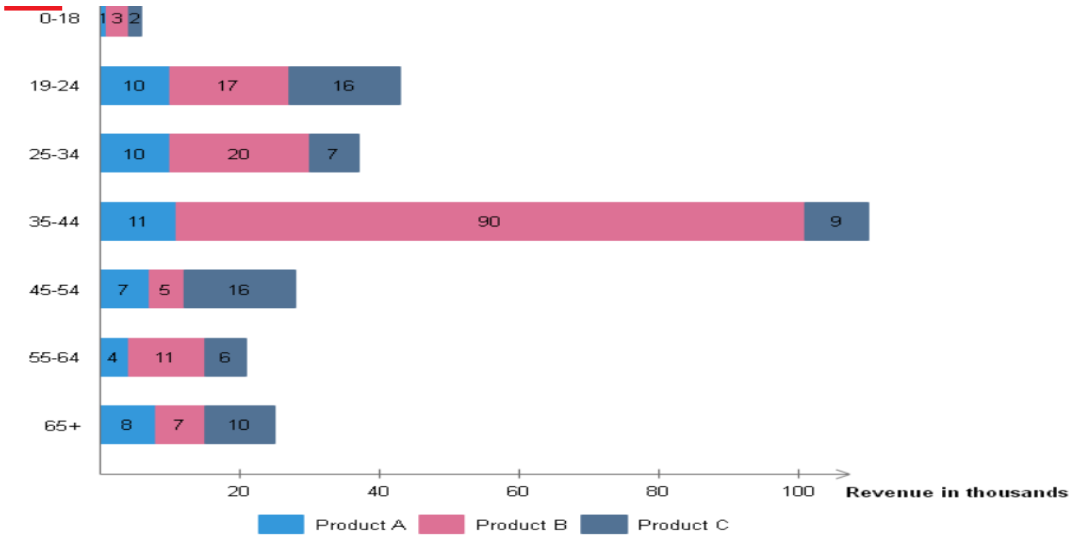
We can easily compare data on each variable at every point of time by the help of bar chart.

As an example, a bar chart might seek to compare the sales of your company in this year to the last year.

What time to use a bar chart?

When it is necessary to compare various categories.

In the event that you have to demonstrate the size of change in data with time.



Column Chart

The column chart is a special kind of bar graph where vertical bars are used to indicate comparison of categories.Whatever can be counted can be represented in column chart.

The column charts are the most appropriate means of depicting the scenario at any given moment (such as the quantity of items sold on a web-shop).

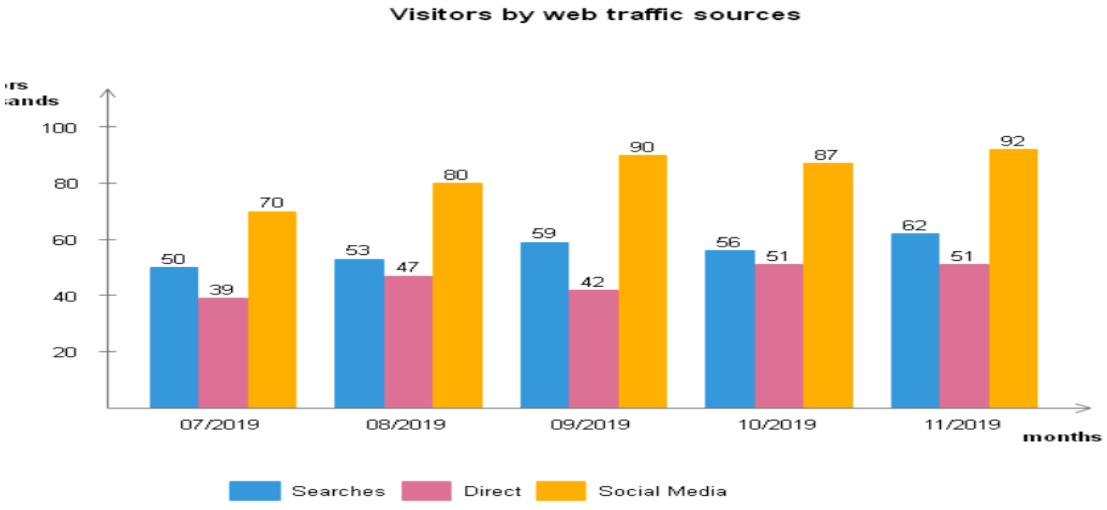
The primary aim of their usage is to focus on the overall numbers as opposed to the trend (trends are better represented by a line chart).

What is the situation to apply a column chart?

You have to demonstrate the comparison side-by-side when you have to compare two different values.

When you would like to underline the distinction between values.

When you wish to point out the cumulative numbers as opposed to the trends.



One of the beautiful types of data visualization is presented in pie charts. On a high level, they are readable easily and they are used in area representational descriptions of relative sizes.

pie chart:

A Pie Chart is a round shaped graph which depicts relativity of sizes of data in terms of pieces of pies.

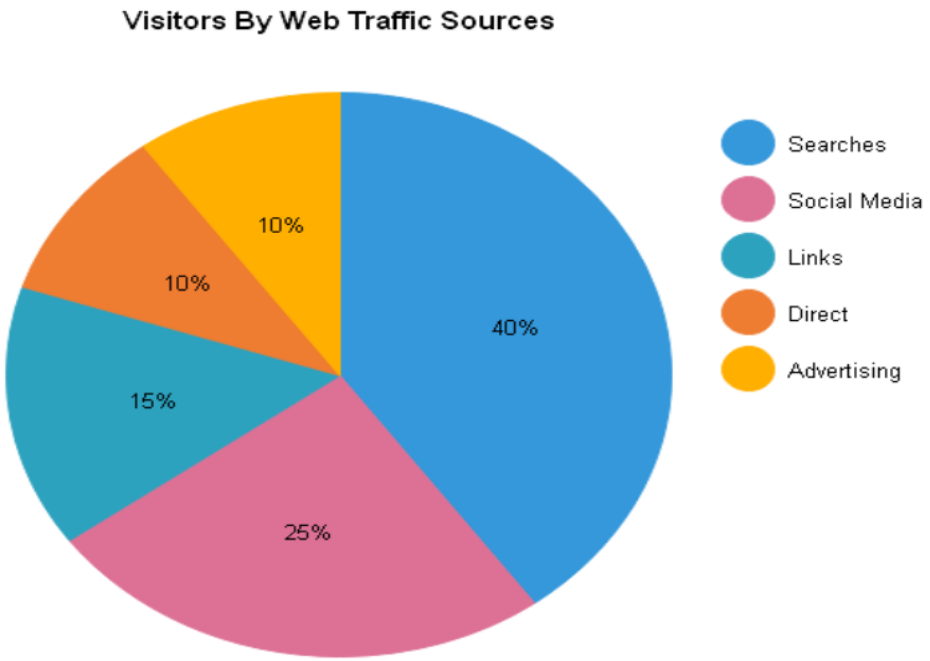
Pie chart is an ideal tool with regard to displaying percentages since it displays each component as a component of the entirety.

The whole pie constitutes 100 percent of a whole. The slices in the pie symbolize parts of the entirety.

When not to use pie chart?

To indicate the percentage that each of the values contributes to the entirety.

When you would like to illustrate the breaking up of a group into smaller parts.



Area Chart:

An area chart is an optimal choice in case you have to represent the data reflecting a time-series relationship.

What is area chart?

An area chart is that kind of a chart that shows the variation of one or more quantities with time. It is like a line graph.

In area charts as well as line graphs, the data values are linked by a line to indicate the value of a quantity at various times. Both of them are good when it comes to displaying trends.

Nevertheless, the area type graphic is not similar to the line graph, as the space between the x-axis and the line is colored. Area charts hence provide a feeling about the total volume.

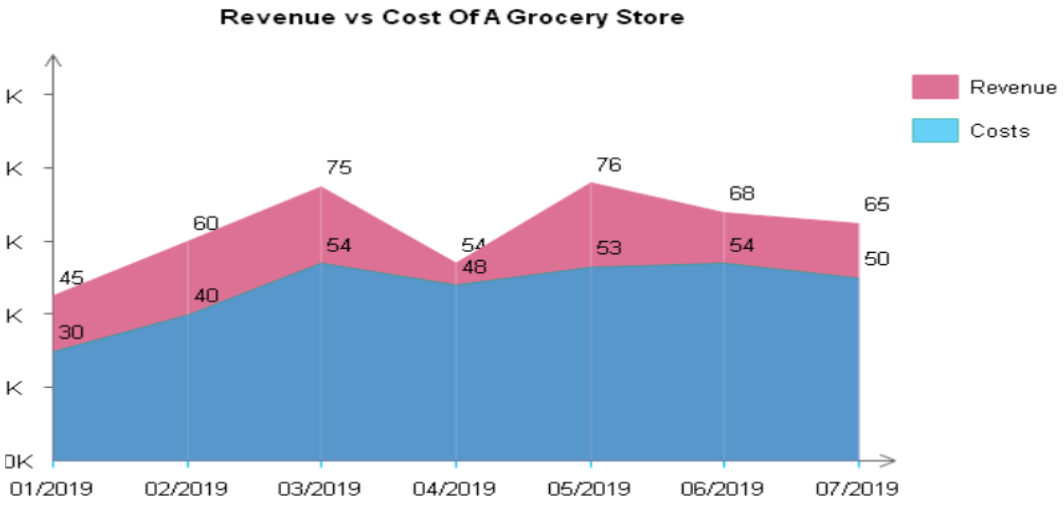
Area charts focus on a trend through time. They do not pay so much attention to a presentation of exact values.

The area charts are also ideal where there is need to exhibit the variation of the different sets of data.

Where to apply an area chart?

In instances where you need to apply a series of lines to draw a comparison between groups (tells aka series).

You need to know the breakdown of the total by groups as well as their whole value when you desire to follow such a value.



Scatter Plot

Scatter plot is also one of the commonly used types of data visualization and it is also referred to as a scatter diagram, scatter graph and correlation chart.Scatter plot is useful in various spheres of the modern world business, biology, social statistics, data science, and etc.

What is Scatter plot?

Scatter plot is a graphical depiction stating a relationship amid two variables. It is aimed at demonstrating the extent of influence that one variable has over another.

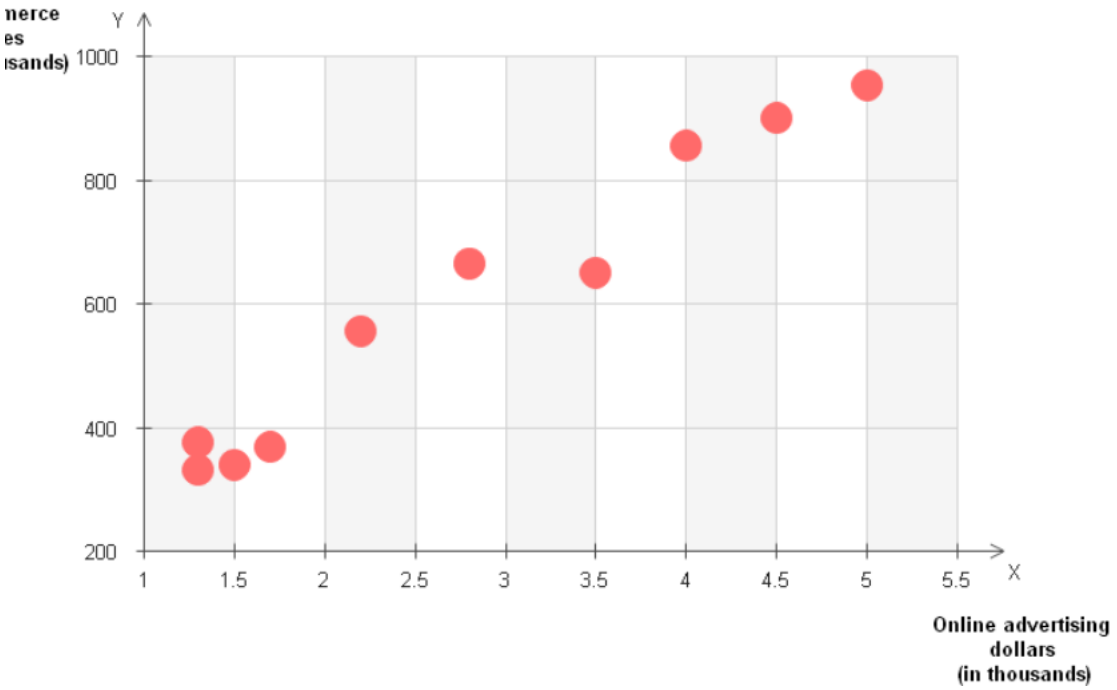
In most cases, in the same formula when there exists an association between 2 variables, the former is referred to as the independent variable. The second variable, as it is known, is called a dependent variable since the values obtainable depend on another variable.

However, there is also a possibility of no kind of relationship between 2 variables.

Usage of a Scatter plot.

You ought to use it when you are required to note and demonstrate relationships between two numerical variables.

When one simply wants to graphically represent the relationship between 2 datasets that are large in nature without relating to time.



Bubble Chart:

The class size paradox

Class size paradox Characteristics Paradox A result of statistics Surprise countParadoxClass sizeA statistical anomalyThe class size paradox is an unexpected statistical occurrence

generally employed in the scope of data science and probability to point out bias in averaging and sampling algorithms. It is in close connection with the sampling bias and the paradox of friends.

Definition:
The class size paradox is the fact that the size of the average class experienced by students is bigger than the size of the average calculated on all classes.

Sampling Bias:
In case you sample individuals instead of groups, your estimates will be biased.

Weighted Averages:
Misapplication of averages which do not take into account weights may result in false conclusion.

Survey Design:
In the analysis of survey or user data (e.g. product usage, class feedback) the assumption that the distribution of participants can be skewed is also valid.

Let's say a school has the following classes:

Class	Number of Students
A	10
B	20
C	30

- **Average class size (per class)**:
  $\frac{10+20+30}{3} = 20$ students.
- **Average class size experienced by students (weighted by students)**:

$$\frac{10 \times 10 + 20 \times 20 + 30 \times 30}{10 + 20 + 30} = \frac{100 + 400 + 900}{60} = \frac{1400}{60} \approx 23.3$$

the average class size is 20, the average *student's experience* is closer to 23.3.

Misleading Metrics:
In business intelligence/A/B testing, results may seem skewed when the data has not been normalized correctly.

Data frame indexing

An efficient way to manipulate data is a top priority in data science, and indexing in DataFrames (as in pandas) can speed up and rationalise how we extract, filter and manipulate data.

What is DataFrame?
DataFrame is a 2 dimensional tabular data structure that has:

Columns (labeled with an index),

Columns (marked with headers),

Just like an excel sheet or an SQL table.

```
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Department': ['HR', 'IT', 'Finance']
}

df = pd.DataFrame(data)
```

Types of Indexing

1. Label-Based Indexing (.loc)
Uses row and column labels (names).

Syntax: df.loc[row_label, column_label]

```
df.loc[0, 'Name']      # Output: 'Alice'
df.loc[0]              # Entire first row
df.loc[:, 'Age']       # Entire Age column
```

2.Integer-Based Indexing (.iloc)
Uses row and column positions (like array indexing).
Syntax: df.iloc[row_index, column_index]

```
df.iloc[1, 0]          # Output: 'Bob'
df.iloc[1]             # Entire second row
df.iloc[:, 2]          # Entire third column ('Department')
```

3. Boolean Indexing (Filtering)
Index rows using logical conditions.

```
df[df['Age'] > 28]     # Returns rows where Age > 28
```

4. Set Index
You can also set a column as an index for faster lookup.

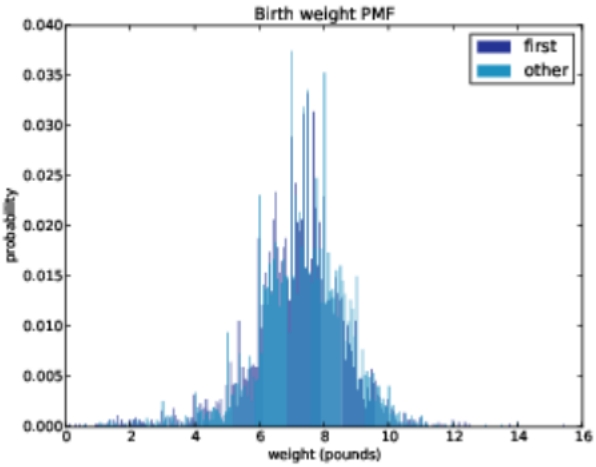
```
df.set_index('Name', inplace=True)
df.loc['Alice']        # Access Alice's row
```

Cumulative distribution functions - Limits of PMFs

Constrains of PMFs

In case the number of values is not large, PMFs will work well. However, the more the values that have to be calculated, the less the probability of each value and the high effect of random noise.

As an example we may be interested in distribution of birth weights. NSFG data Total wgt lb variable stores the weight of a baby at birth, in pounds. The PMF of this quantity of first babies and other is given in Figure below.



PMF of birth weights

In general, such distributions take the shape of a bell of a normal distribution, where the values are higher along the mean and far apart in both ends.

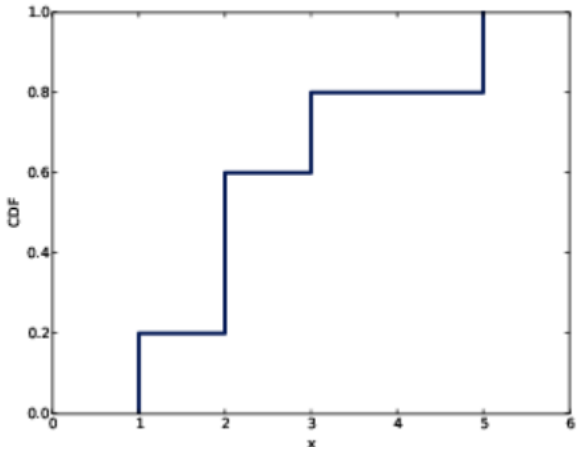
However, components of this figure are not easy to read. Numerous spikes and valleys are present and there are some differences that can be considered visible between the distributions. Which of these features are meaningful, it is hard to tell. Too, it is difficult to observe total trends; e.g., which variance is it that you think has the larger average?

Binning of the data can overcome these problems; i.e., putting data range into non-overlapping (binned) intervals and tabulating the number of values within each bin. Binning may be helpful but bin size is something difficult to correct. By smoothing out the noise they

may also smooth out the useful information, too, should they be large enough.

```
def EvalCdf(sample, x):
    count = 0.0
    for value in sample:
        if value <= x:
            count += 1

    prob = count / len(sample)
    return prob
```



This is essentially equivalent to PercentileRank, but the output stands in the interval [0,1] instead of having the percentile rank in the interval [0,100].

To make an example, consider the sample to be taken as [1, 2, 2, 3, 5]. These are some of its CDF values:

CDF(0) = 0
The CDF (1) = 0.2
The CDF(2) = 0.6
The CDF(3) = 0.8
CDF(4) = 0.8
The likelihood that five goals are scored is 1
We are in a position to assess the CDF of any x and not necessarily the values that are present in the sample. CDF(x) is zero when x is less than minimum in the sample. In the case of x that exceeds the largest value, CDF(x) is 1.

Representing CDFs

Cdf basic procedures offered are:

Prob (x): for a given value x, the probability is calculated as p = CDF (x). The bracket operator is synonymous to Prob.
Value(p): Computes the value, x corresponding to a probability, p; i.e. the inverse CDF of p.

The constructor of Cdf accepts the argument of a list of values, a pandas Series, a Hist, Pmf or another instance of Cdf. The next listing creates a Cdf of the distribution of the length of pregnancy in the NSFG:

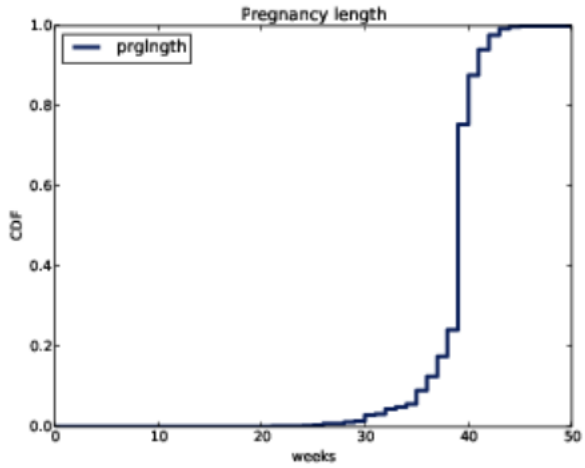


Figure 4.3: CDF of pregnancy length.

```
live, firsts, others = first.MakeFrames()
cdf = thinkstats2.Cdf(live.prglngth, label='prglngth')
```

thinkplot has a function Cdf which graphs Cdfs as lines:

```
thinkplot.Cdf(cdf)
thinkplot.Show(xlabel='weeks', ylabel='CDF')
```

Percentile based statistics

Percentiles refer to the proportion of the scores which lie below a given score. They are a sort of quantile which divides the data into 100 roughly equal sized cell and informs you the location of a score relative to other scores. To illustrate this, somebody who has an IQ of 120 is on 91 st percentile indicating that his or her IQ is greater than 91 other scores.

When you want the relative position of a number, percentiles are a fine thing to have.
Percentiles give you an idea about how a value stands against other values. The simple rule is that where X appears at kth percentile then, X exceeds K percent of the values. Now, let us view how this information may be useful.
. As a result there are various ways of computing percentile. This post lists four procedures. The first three are the practices that analysts resort to in finding the percentiles when examining the actual values of data in moderately sized data sets. The k th percentile is defined differently by the above three definitions as below:

The lowest value that exceeds more than k-per cent of the values.
The least amount that supersedes or equals to k percent of values.
A computed value between two nearest ranks.

Random numbers

The importance of random numbers in data science Researchers, amateurs, and students use random numbers to create such models as random sampling and generate synthetic data and initialize machine learning models.
Random numbers are values to which a number is created so that all the numbers are likely to be chosen equally.

Application	Purpose
Data Splitting	Randomly divide data into training/testing sets
Bootstrapping & Resampling	Create new datasets from samples with replacement
Simulations (e.g., Monte Carlo)	Run repeated random experiments to estimate outcom
Synthetic Data Generation	Create artificial data for testing models
Random Forests & ML Algorithms	Use randomness to increase robustness and reduce overfitting

They are typically pseudo-random in data science explains in data science, the algorithms are typically pseudo-random (i.e. not really random yet sufficiently random to be useful in practice).

Generating Random Numbers:

```
np.random.randint(1, 10, size=5)
# e.g., [7, 2, 4, 9, 1]
```

```
np.random.rand(5)
# e.g., [0.78, 0.12, 0.45, 0.89, 0.33]
```

```
np.random.normal(loc=0, scale=1, size=5)
# e.g., values centered around 0 with standard deviation 1
```

```
data = np.array([1, 2, 3, 4, 5])
np.random.shuffle(data)
```

Comparing percentile ranks:

Percentile ranks are a strong statistical concept in data science and they compare values with reference to a distribution. Two percentiles represent the proportions of observations shown by a dataset that are less than a particular value.

Suppose a student scored at the 90th percentile, or above 90 percent persons and he is better than them.

```python
from scipy import stats
import numpy as np

data = [55, 60, 65, 70, 75, 80, 85, 90, 95, 100]
value = 80

percentile_rank = stats.percentileofscore(data, value)
print(f"{value} is at the {percentile_rank}th percentile.")
# Output: 60.0
```

Comparison of Percentile Ranks:

```python
math_scores = [55, 60, 65, 70, 75, 80, 85]
science_scores = [60, 62, 65, 68, 70, 72, 75]

math_score = 70
science_score = 68

math_percentile = stats.percentileofscore(math_scores, math_score)
science_percentile = stats.percentileofscore(science_scores, science_score)

print(f"Math: {math_percentile}th percentile")
print(f"Science: {science_percentile}th percentile")
```

Modeling distributions - Exponential distribution

Its probability distribution is used when modelling the time between independent events in a Poisson process, in data science and in statistics (i.e., when events occur one at a time at a constant mean rate).

- Exponential distribution is applicable in modelling the time to event.
- It has the assumption that events are independent and occur at random.
- The rate of occurrence = λ (lambda).
- Unlike other continuous distributions it is memoryless.

Feature	Description
Shape	Right-skewed (decays exponentially)
PDF	$f(x;\lambda) = \lambda e^{-\lambda x}$ for $x \geq 0$
Mean	$\frac{1}{\lambda}$
Variance	$\frac{1}{\lambda^2}$
Memoryless Property	The future is independent of the past

Use Case	Example
Queueing systems	Time between customer arrivals at a service counter
Reliability analysis	Time until failure of a machine or component
Web analytics	Time between website hits or API calls
Survival analysis	Lifespan of systems, organisms, etc.
Simulation	Synthetic modeling of real-world events

Modeling Exponential Distribution in Python

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import expon

# λ = 1.5 (rate parameter)
lambda_val = 1.5
data = np.random.exponential(scale=1/lambda_val, size=1000)

plt.hist(data, bins=30, density=True, alpha=0.6, color='skyblue', label='Histogra

# Plot PDF
x = np.linspace(0, max(data), 1000)
plt.plot(x, expon.pdf(x, scale=1/lambda_val), 'r', label='Exponential PDF')

plt.title('Exponential Distribution (λ=1.5)')
plt.xlabel('Time')
plt.ylabel('Probability Density')
plt.legend()
plt.show()
```

Normal distribution

The Gaussian distribution also referred to as the normal distribution is a bell-shaped probability distribution that is symmetrical about the mean.

It is one of the most significant distributions in statistics and data science since there were many natural and social phenomena that this distribution tends to follow.

Feature	Description
Shape	Bell-shaped, symmetric
PDF (probability density function)	$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
Mean (μ)	Center of the distribution
Standard deviation (σ)	Controls the spread (width of the curve)
Empirical Rule	~68% within 1σ, ~95% within 2σ, ~99.7% within 3σ

Use Case	Example
Error distribution	Residuals in regression often assumed to be normal
Statistical inference	Many tests (t-test, z-test) assume normality
Standardization (Z-scores)	Used to compare different scales
Data normalization	Feature scaling for ML algorithms
Anomaly detection	Values far from the mean are flagged as anomalies

Implementation:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
  mu = 50      # mean
  sigma = 10   # standard deviation

  # Generate random data
  data = np.random.normal(loc=mu, scale=sigma, size=1000)
```

```
# Plot histogram
plt.hist(data, bins=30, density=True, alpha=0.6, color='skyblue', label='Data')

# Plot the PDF
x = np.linspace(mu - 4*sigma, mu + 4*sigma, 1000)
plt.plot(x, norm.pdf(x, mu, sigma), 'r', label='Normal PDF')

plt.title('Normal Distribution (μ=50, σ=10)')
plt.xlabel('Value')
plt.ylabel('Probability Density')
  plt.ylabel('Probability Density')
  plt.legend()
  plt.grid(True)
  plt.show()
```

Lognormal distribution.

P probability distribution -A lognormal distribution is a continuous probability distribution of a random variable, such that the logarithm of the variable follows a normal distribution.

Feature	Description
Shape	Right-skewed, non-negative
Support	$X > 0$ (values are always positive)
PDF	$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}$
Mean	$e^{\mu+\sigma^2/2}$
Median	e^{μ}
Mode	$e^{\mu-\sigma^2}$

Use Case	Example
Income distribution	Most people earn below average; few earn extremely high
Stock prices	Prices cannot be negative, and changes are multiplicative
Biological measures	Weight, growth rates, or reaction times
Time to failure	When failure rate increases over time

Implementation:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import lognorm
```

```python
mu = 0       # Mean of log values
sigma = 0.9  # Std dev of log values

# Generate data
data = np.random.lognormal(mean=mu, sigma=sigma, size=1000)

# Plot histogram
plt.hist(data, bins=30, density=True, alpha=0.6, color='skyblue', label='Histogra

# Plot the PDF
x = np.linspace(0.01, max(data), 1000)
pdf = lognorm.pdf(x, s=sigma, scale=np.exp(mu))
plt.plot(x, pdf, 'r', label='Lognormal PDF')

plt.title('Lognormal Distribution (μ=0, σ=0.9)')
plt.xlabel('Value')
plt.ylabel('Probability Density')
plt.legend()
plt.grid(True)
plt.show()
```

Module 5

Time Series Analysis And Predictive Modeling

Time Series Analysis refers to a way of data analysis by considering the points after a fixed time period (e.g.; daily, monthly, annual) in order to derive certain pattern, trends to construct forecasts.

Time series analysis and forecasting play the most important role in predicting future trends, behaviours and behaviours using the past data. It assists companies to make informed decision, efficient resource allocation, as well as avoid risks, through its ability to predict market demand, sales volatility, stock changes, and so many others. It also helps to plan, budget and strategize in different fields including finance, economics, health, climatic science, and resource usage and promotes efficiency and competitiveness.

Properties of Time Series Data:

Temporal Order: The information is sequenced with time (chronologically).

The observations rely on time (a past observation affects other future observations).

Regular Intervals: The data are taken regularly (i.e., hour-to-hour, day-to-day, month-to-month).

The data points are given as the result of watching or recording something in time, like the price of a stock, the temperature in a room, or the amount of sales being made. Time series data Time series data is typically displayed graphically, with time represented on an axis and the variable of interest on an axis so analysts can determine trends, patterns, and time changes.

The representation of the time series data is commonly performed in the form of a line plot where the time is shown on the horizontal axis (x-axis) and the values of the variable on the vertical axis (y-axis). The graphical representation allows the visual interpretation of trends, patterns and fluctuations in the variable over time to assist in the analysis and interpretation of the data.

Importance of Time Series Analysis

Forecast Future: Time series analysis can be used to derive the future trend and modeling future behavior of businesses in order to help them to be proactive in their decision making such as anticipating the market need, stock prices and other variables.

Identify Patterns and Anomalies: Time series analysis can also aid in identifying repeat patterns and outliers by scrutinizing series of information points which render insight into underlying dynamics and possible outliers.

Risk Mitigation: Through the identification of risks, the businesses are able to come up with mitigation strategies to deal with risks, and this plays a significant role in the management of risks.

Strategic Planning: Behaviors and patterns derived out of time series help in long term strategic planning, thus making decisions about what to do in the finance world, healthcare and other sectors.

Competitive Advantage: Time series analysis helps companies to better allocate resources both with regard to inventory, staff and monetary resources. Businesses can make decisions based on data and be ahead of the market trends and react to changes to gain a competitive advantage.

Elements of Time Series Analysis

Trend: Whereas a non-uniform interval does not exist and any deviation of the given set of data is any constant flow of time. The trend would be negative or positive or null trend

Seasonality: That is some repetitive or deterministic period change in the data on an ongoing time-line. Would be Bell curve or saw tooth

Cyclical: Under which there is no specific time, a lack of certainty in the direction and trend of movement

Irregularity: The sudden occurrence/event/scenario and spikes within a brief period of time.

	Trend	Seasonality	Cyclical	Irregularity
Time	Fixed Time Interval	Fixed Time Interval	Not Fixed Time Interval	Not Fixed Time Interval
Duration	Long and Short Term	Short Term	Long and Short Term	Regular/Irregular
Visualization				
Nature - I	Gradual	Swings between Up or Down	Repeating Up and Down	Errored or High Fluctuation
Nature – II	Upward/Down Trend	Pattern repeatable	No fixed period	Short and Not repeatable
Prediction Capability	Predictable	Predictable	Challenging	Challenging
Market Model				Highly random/Unforeseen Events – along with white noise.

Types of Time Series Analysis

Since time series analysis involves numerous categories or versions of data, there is a situation where an analyst has to construct a complicated model. Nevertheless, not all the variances can be explained by analysts, and they are not able to generalize a particular model to all samples. Too complex models or models that attempt too much are also a fit problem. Failure to fit (lack of fit) or failure to partition random error and true relationship (overfitting) models means that models fail to draw the distinction whereby analysis is skewed and forecasts are inaccurate.

Examples of time series analysis models are:

Classification: Investigates and allocates patterns to the data.

Curve fitting: Graphs the information in a curve to analyze the relations of variables in the data.

Descriptive analysis: Determines the trends, cycles or seasonal variation in the data of time series.

Explanative analysis: Tries to grasp the information and internal relations in it, cause and effect.

Exploratory analysis: Displays the key features of the time series data, typically in graphic form.

Forecasting: It guesses values to come. This is the trend-type which is historically oriented. It applies the past in modeling the future data, forecasting of situations that may occur in future plot points.

Intervention analysis: examines how data can be altered by an event.

Segmentation: Divides the data into portions to demonstrate the property of source information.

Data classification

Time series data can be identified with two broad categories:

The data on **stock times series** implies that the values of the attributes are measured at a defined time, such as a fixed picture of the data as it was during the time.

Flow time series data entails a record of the activity of the attributes within a specified duration which is usually a component of the whole and forms a section of the outcomes.

In a time series, the variations may appear randomly in data:

Through **functional analysis**, one will be able to select patterns and relationships in data so as to find significant events.

Trend analysis refers to ascertaining uniform change at a particular direction. The two classifications of trends are the deterministic and stochastic trends; the latter is random and cannot be explained, and the former can be traced to its ultimate cause.

Seasonal variation refers to occurrences of certain and periodic times in the period of one year. Serial dependence is a situation where subsequent points in time are likely to be linked.

Time Series Forecasting

Time Series Forecasting refers to a statistical time series method that is able to provide pointer to future series using past series values. Simply put, thinking of it is gauging into the future of data value in time. Time Series Forecasting can be used to Figure out future trends using trends and patterns identified in past records and help in decision-making and planning by leading to educated guesses concerning what is likely to occur.

Various Time Series Forecasting Algorithms

Autoregressive (AR) Model: Autoregressive (AR) model is a model included in the time series with the help of which further values can be predicted on the basis of linear combinations of prior values of the same time series. AR(p) A time series is modeled in an AR(p) model as a linear combination of its past p values, plus a random error term. Autoregressive model Because it is the order of the models (p), there is the number of previous values that are included in the prediction process.

Autoregressive Integrated Moving Average (ARIMA): ARIMA models are popular models of time series forecasting. It estimates the following term in a time series with the combination of its past values as well as the previous forecast errors. The parameters of the model are order of autoregression (p), differencing (d) and moving average (q).

ARIMAX: extension of ARIMA to incorporate exogenous variables that may result in a more accurate set of forecasts.

Seasonal Autoregressive Integrated Moving Average (SARIMA): SARIMA is an extension of ARIMA where seasonality is accounted in the model. It has extra seasonal parameters (P, D, Q) to repay periodic changes in the data.

SARIMAX: An augmentation of SARIMA used with seasonal time series forecasts that uses exogenous variables.

Vector Autoregression (VAR) Models: VAR models are autoregression of multivariate time series data where each of the variables is a linear combination of its own past values and the past values of the other variables. They become applicable when there is a need to examine and predict interdependencies between several time series.

Theta Method: The most basic and the obvious forecasting method that is obtained through extrapolation and trend approximation.

Exponential Smoothing Techniques: Exponential smoothing techniques, according to the names Simple Exponential Smoothing (SES) and Holt-Winters, predict the upcoming values with exponentially declining weighting on historical values. Such approaches are especially helpful when dealing with trended and seasonal data.

The Gaussian Processes Regression: Gaussian Processes Regression is another Bayesian non-parametric method of modeling functions distributions through time. It gives the uncertainty estimates together with point forecasts, hence it is applicable in capturing the time series forecast uncertainty.

Generalized Additive Models (GAM): is a general modeling tool that uses additive components, which makes it possible to have a nonlinear relationship and interactions.

Random Forests: Random Forests is a method of related machine learning where multiple decision trees are grown during training and the average of the predictions of the multiple trees are returned. It is suitable to forecast using time series as the data can contain complicated relationships and interaction between the data.

Gradient Boosting Machines (GBM): GBM represents the next ensemble-based learning method which by sequentially constructing several decision trees, where each one can rectify the more recent mistakes of the preceding tree. It has excellent performance and has high resistance to overfitting in determining nonlinear relationships.

State space models: These state space models are a description of a time series as a mixture between unobserved (hidden) states and observed measurements. These models reflect in

reference to the deterministic and stochastic part of the time series, which means that they can be used in forecasting and observation of any anomalies.

Dynamic Linear Models (DLMs): DLMs are a state-space Bayesian model that use a model of time series data which is a representation of a latent state. They are flexible in that they can absorb different trends, seasonality and other moving trends in the data.

Recurrent Neural Network (RNNs) and Long Short-Term Memory (LSTM) Networks: RNNs and LSTMs allow working with sequential data, which is another deep learning structure. They are able to learn intricate time-varying patterns in time series data and hence are effective in forecasting problems particularly on large scale and high dimensional data.

Hidden Markov Model (HMM): An Hidden Markov Model (HMM) is a mathematical model that describes the sequential results of some observable events as a combination of some hidden states. HMMs enable the inference of unobservable states with respect to observed data within time series and allow defining the dependency and state changes between states. They find useful application in such activities as speech recognition tasks, gesture analysis, and anomaly detection, where they present a framework to learn complex sequential data and discover patterns within that data, leading to meaningful constructions of the data.

Importing and Cleaning Time Series Data

Importing:

Read CSV, Excel, databases or APIs using libraries such as Pandas.

```
import pandas as pd

df = pd.read_csv('timeseries.csv', parse_dates=['Date'], index_col='Date')
```

Cleaning:

Treat missing items, duplications, wrong timestamps.

Make datetime an index of the DataFrame.

submit data into proper frequencies (e.g. daily, monthly) with .asfreq() or resample().

Plotting

The graphical representation of data that has been collected over time intervals forming a chain is the time series visualization. It includes a wide range of such methods as line plots, seasonal subseries plots, autocorrelation plots, histograms and interactive visualization. The techniques are useful in enabling the analysts to detect trends, patterns, and anomalies in time-varying data in order to understand and make better decisions.

Various graphs of visualization of Time series

Line Plots: The line plots are plotted based on a trend, a cycle, and changes in data patterns, which can be easily observed.

Seasonal Plots: Seasonal plots partition time series data into seasonal and non-seasonal data; this enables one to examine trends over a particular period of time.

Histograms and Density Plots: Plots the values of the data and the time of occurrence, thus giving us ideas about the nature of data such as skewness and kurtosis.

Autocorrelation and Partial Autocorrelation plots: Plots to help discover seasonality and lagged relationships as well as time series autocorrelation.

Spectral Analysis Spectral analysis tools, like periodograms and spectrograms, represent the frequency components of a time series data and can be valuable to detect periodicity and cyclical trends.

Decomposition Plots: Decomposition plots separate a time series into its trend, seasonal, and residual prices in order to determine patterns underlying a data set.

Use **Matplotlib** or **Seaborn** for line plots.

```
import matplotlib.pyplot as plt
df['Value'].plot(title='Time Series Plot')
plt.show()
```

Moving averages

Traders resort to different indicators to assist in making buy and sell decisions in technical analysis. The moving average is one of such indicators that can be utilized to smooth movements in prices on a daily basis to aid in developing a trend on the daily pricing by developing an average price that remains continuously updated.

The effects of short-term random fluctuations in the price of a stock on a given period of time can be reduced by computing the moving average. Simple moving averages (SMAs) incorporate the use of a simple arithmetic average of prices over this period of time whereas the exponential smoothing (EMAs) applies heavier weightings on more recent prices than old prices over the timespan.

Calculation of Moving averages is done to determine the direction of a trend of a stock or to prefer its support and resistance values. It is also trend or lagging indicator since it is pegged on the historical prices.

- Irregularities the short returns, and emphasizes the tendencies.

• Types:

Simple Moving Average (SMA):

A simple moving average (SMA) is determined by the arithmetic mean of a certain the set of values over a period of time. A tranche of figures or prices of stocks are added and the sum is divided by the number of prices in the set. A security has an average security value that can be calculated using the following formula:

```
df['SMA_3'] = df['Value'].rolling(window=3).mean()
```

$$SMA = \frac{A_1 + A_2 + \ldots + A_n}{n}$$

where:

$A =$ Average in period n

$n =$ Number of time periods

Exponential Moving Average (EMA):

The exponential moving average puts larger weights on recent prices as this is supposed to make them adapt more easily to new information. In order to compute an EMA the simple moving average (SMA) over certain time period is computed.

Then take the multiplier coefficient which scales the EMA, or so-called the smoothing factor, which is usually, [2/(chosen time period + 1)].

```
df['EMA_3'] = df['Value'].ewm(span=3).mean()
```

$$EMA_t = \left[V_t \times \left(\frac{s}{1+d}\right)\right] + EMA_y \times \left[1 - \left(\frac{s}{1+d}\right)\right]$$

where:

$EMA_t =$ EMA today

$V_t =$ Value today

$EMA_y =$ EMA yesterday

$s =$ Smoothing

$d =$ Number of days

Missing values

Missing values or the data that is not (or not being) stored (or not being present) of some variable/s in the given dataset is called missing data.

Value that is missing in a data may be recorded in many forms, depending on the origin of the data and the conventions. Some of the typical representations here are:

NaN (Not a Number): Missing data is expressed as NaN in a lot of programming languages and data analysis packages. That is the default library of libraries such as Pandas in Python.

None or NULL: Different programming languages and databases usually have missing values, which are marked as None or NULL. An example is SQL databases where a missing value is supposed to be noted using NULL.

The values or data that are not stored (or not there) to some variable/s in the given dataset are called the missing data.

In some databases missing values may be marked by a special symbol, in others a given code can be used which reflects that there is no current value. These are some typical representations:

There might be various explanations in the absence of validations of particular values in the data. The cause of data missing on the dataset influences the process of missing data processing. Therefore, there is the need to learn what might be the reason why the data might be absent.

Among some of them are the reasons below:

Information stored in the past may be affected by failure to maintain.

There are no observations done in some of the fields because of some reasons. This may lead to inaccuracy of recording of the values since it works through humans.

Intentional failure of the user in providing the values is not provided

Item nonresponse: It is an indication that the participant did not participate.

Missing Values: In some cases, the symbols used to represent a missing value include empty strings (""). This usually happens in the data that is given in text format or CSV where a field may be empty.

Special Indicators: Datasets may consider these specific values to represent special indicators, such as -999, 9999, or other values that are unlikely to occur, indicating that it is a special value, i.e. that the data point is missing. Many times it can be observed with older datasets or in certain industries where such conventions have been provided.

Blanks or Spaces: Missing values could also be indicated by spaces or blank fields in some instances especially in the text files whose widths are fixed**.**

- Strategies:
    - **Forward fill**: `df.fillna(method='ffill')`
    - **Backward fill**: `df.fillna(method='bfill')`
    - **Interpolation**: `df.interpolate(method='linear')`
    - **Dropping rows**: only if missing data is minimal
- **Visualize** missing data using `missingno` or `seaborn.heatmap`.

Serial correlation

Serial correlation Within a time series, serial correlation is the property of a variable and a lagged version of the same variable (i.e. a variable at time T and a variable at time T-1) being found to be correlated with each other over time. Serial correlation is common in repeating patterns when the level of one variable influences future level of the variable. The technical

analysts employ this correlation in financial terms to know how effectively the values of the prices of a security in the past are able to predict the future of the price of the same security.

Serial correlation is analogous to the statistic lagged or auto correlation.

Serial correlation can be defined as a statistical term used to explain the correlation that exists amidst the observations of identical variable regarding particular periods. When the serial correlation of a variable is found at zero then there is no correlation, and all the observations are independent of the other. On the other hand, when the serial correlation of one variable is biased towards 1, then the observations are serially correlated and prior observations influence the outcome of the future observations. In essence, serial correlation variable is not random but has a pattern.

Serial correlation as applied to engineering has the purpose of measuring how a signal, (eg, a signal sent out by a computer or a wave in the radio band) varies relative to itself as a function of time. This idea increased its popularity in economic circles where economist and econometricians used the measure to compare economic data over periods of time.

Nearly every major financial organization currently employees quantitative analysts, quants, to aid in mathematical analysis of some aspect of their business. Technical analysis and other statistical inferences are what these financial trading analysts apply in analyzing and forecasting on the stock market. With the aim to make better predictions and opportunities of a strategy being successful in terms of profitability these modelers are trying to determine the form of the correlations. Also, determining correlation structure more accurately enhances the reality of any of the simulated time series to be based on the model. Precise simulations decrease the danger of the investments strategy.

The success of many of these financial institutions depends on quants because they offer the institution market models which in turn constitutes a foundation of investment affair of the institution.

Autocorrelation

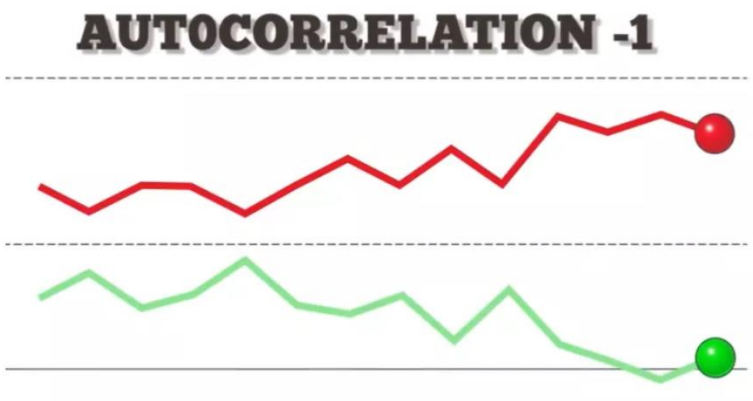
Autocorrelation A mathematical description of the extent to which a particular time series resembles a lagged realization of the same series with respect to successive time span. It is analogous to the correlation between two different time series but the two time series used in autocorrelation are the same but one series is lagged one or more time periods.

To give an example, assuming that it is raining today, the data will indicate that there is a higher probability that it will rain tomorrow as compared to today when it is clear. In the case of investments, equity is the stock that may exhibit powerful positive autocorrelation of returns to indicate that when the stock is up today, it is much more probable that it would be up tomorrow as well.

Of course, autocorrelation may be an effective concept that can be applied by traders; especially by technical analysts.

Autocorrelation is also called lagged correlation and serial correlation since it evaluates how a current value of a variable relates to its preceding values. Just to give a very simple illustration and look at the five values in the chart below in percentages. These we compare them with the column on the right, which has identical set of values, simply shifted up one row.

Day	% Gain or Loss	Next Day's % Gain or Loss
Monday	10%	5%
Tuesday	5%	-2%
Wednesday	-2%	-8%
Thursday	-8%	-5%
Friday	-5%	



Predictive modeling – Overview

Predictive modeling is a mathematical process that is employed in the prediction of the future by identifying the trends in a particular and available input data. It forms part of a very important element of predictive analytics, a form of data analytics that is used to predict activity, behavior and trends by using both current and historical data.

Computing the quality of a sales lead, the probability of spam or the probability a person will click on a link or purchase a product are examples of a predictive models. The functionality is frequently embedded within various business applications thus it may be worth getting to know the mechanics of predictive modeling operations in order to do troubleshooting and optimize its functionality.

Despite the implication of the term predictive modeling (to be concerned with the prediction of the future), it can be used to make predictions (e.g., the likelihood that a transaction is fraudulent). Here the act of fraud has already taken place. The idea, here is to forecast that the transaction will be considered as fraud in the future analysis. The predictive modeling is also able to predict future requirements or what-if analysis.

Evaluating Predictive models

The current data-driven solution engines are the machine learning models, and the question is how would one know whether they are performing their duties, i.e. working well? The model evaluation metrics would play this role: assisting a person in grasping the strengths and weaknesses of a model with the aim of optimization and final implementation of the matter in real life. Whether it is a model to store which emails should be classified, house price forecast, customer segment clustering, among others, learning how to assess a model based on different metrics is a valuable job.

A confusion matrix is a visualization of prediction performance by categorizing outcomes:

- **True Positive (TP)**: Correctly identified as positive
- **True Negative (TN)**: Correctly identified as negative
- **False Positive (FP)**: Incorrectly identified as positive
- **False Negative (FN)**: Missed positive cases

#	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Accuracy, defined as (TP+TN)/(Total), is simple. However it can be misleading in an unbalanced dataset.

$$\text{ACCURACY} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision: Also known as Positive Predictive Value, it is the value that shows the true positive cases calculated against a total number of predicted positive outcomes.

$$\text{PRECISION} = \frac{TP}{TP + FP}$$

Recall (Sensitivity or True Positive Rate): It measures the percentage of the true positive out of all the real positives.

$$\text{RECALL} = \frac{TP}{TP + FN}$$

F1 Score: It measures a range between precision and recall, and gives an equal balance to the trade-off between precision and recall.

$$F1 = \frac{2 \times \text{PRECISION} \times \text{RECALL}}{\text{PRECISION} + \text{RECALL}}$$

ROC-AUC (Receiver Operating Characteristic -Area Under Curve)

The ROC curve plots the relationship between False Positive Rate (FPR) and True Positive Rate (TPR) at each of different threshold levels of classification. The AUC (Area Under Curve) condenses the capacity of the model to differentiate between the classes:

AUC 1.0: The ideal classifier

ABC

AUC= 0.5: Random accuracy

Regression Metrics: In predicting continuous values

Root Means Error squared (RMSE): Standard deviation of residuals:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Mean Absolute Error (MAE): Simplicity at Work

MAE is the average of the absolute differences between the predicted and actual value has been calculated.

$$\text{MEAN ABSOLUTE ERROR} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE)

$$\text{MEAN SQUARED ERROR} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|^2$$

As RMSE without the square root:

$$\text{ADJUSTED R}^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

Building Predictive Model solutions

Predictive analytics has received the status of a separate area within the scope of data analysis, which employs a range of tools, including statistics, machine learning, and artificial neural networks towards a possible forecasting of the future course of events, relying solely on the past data.

The primary objective of predictive analytics is expressing what could occur in the future, such as trends, particular behavior and events, amongst others, to automatically classify data points, or anticipate possible outcomes with data. In simple terms, predictive analytics involves utilizing all the information in determining the probability of occurrence.

In addition to the above, there are other forms of analytics that you have probably heard of, i.e. descriptive analytics, diagnostic analytics, and prescriptive analytics, which are simply the levels of analytics maturity.

However, before immersing yourself into data and algorithms, it is crucial first to determine the kind of issues you wish to overcome through the predictive analytics model. This could be through a consultation with stakeholders who you are looking to hear about their frustrations, and what their expectation and success would be like.

Data collection

After having defined the goals, you continue to gather information to generate a model of a predictive analysis. In the case of data, most of the time it falls into 2 categories:

Scenario 1 -Your company already has sufficient internal business data that has been accumulated over the years.

Scenario 2 - There is no data at your disposal or the amount of it is not copious. In case it is so, it will be required to include data based on external sources (e.g. free public datasets; some data can be obtained on providers).

Processing and preparation of data

The most crucial and lengthiest stage of the AI predictive modeling segment which can be described as data preparation and processing. The stage may consume as much as 80% of the total project time since detailed work is involved.

Data cleaning

The step to take is to eliminate mistake and discrepancies. This is in the sense that we:

remove noise

correct inaccuracies

Data that are blanks can be replaced by average numbers

delete un finished or abnormal records, etc.

Clean data will help to eliminate low-quality inputs that may distort predictions of the model.

Data transformation

Once cleaned the data must often be converted into an appropriate model-ready format. In this step we approach things in the following way.

Feature engineering

It is the method of applying domain knowledge to filter, transform or construct new characteristics out of raw data so as to aid the predictive capacity of machine learning algorithms. In essence, all we evaluate is the usefulness of the current available data features and concur to determine whether new data features are required or not.

This will need that a deep analysis of the data is carried out to find the latent correlations and it will identify the importance of every feature.

To illustrate, we may not use height and weight as two different features when manipulating the data of patients, but we form a new variable, known as Body Mass Index (BMI).

Training and test set generation

The last procedure is to split the information into distinct groups: traditionally 80 percent will be used to train and 20 percent to test and verify. Such division is used to train the model correctly and measure its performance properly.

Choicing of predictive modelling strategies

When we possess some data we choose some predictive modeling method. Different predictive models can be selected, depending on a task you want to solve. And, as the example it is possible to put before us Scikit-learn library where even in the separate roadmap the idea of what model better to choose according to what and how much data is already at hand and what objective is can be grasped.

Classification

The classification of machine learning is merely the forecast of group or category of the data under consideration. It receives input data, and a category (class) label is applied to these data depending on the features they have.

nce we select the appropriate future model, we train it using your ready dataset. During this phase, you set a specific parameter - the configuratory elements that determine how the model behaves. For example, a healthcare model that predicts the risks of lung cancer can control the rate of learning or model to prevent overfiting.

We also need to adjust these parameters to improve the accuracy and efficiency of the model. For example, in e-commerce applications for predicting demand, these adjustments may be favorable for handling seasonal trends or changes in consumer behavior.

In addition, when we train a future stating model, we assess its performance against testing data using matrix that are suitable for its type.

For classification tasks, we use matrix such as:

The accuracy assesses the overall purity of the model.

The accurate checks how many examples of the model predict correctly.

In regression functions, normal matrix includes:

The mean paid error (MSE) calculates the average repayment difference between estimated values and real values.

R-Squar shows how well the model predicts unseen samples.

Of course, there are many other machine learning matrix to include.

Sentiment Analysis

The emotion analysis, also known as opinion mining, involves subjective information such as processing text data to remove emotions, opinions or approaches. This enables machines to explain the spirit of the text material by analyzing keywords, references, tones and grammar.

For example, analyzing product reviews to determine that customers are satisfied or dissatisfied, it is a general use of emotion analysis.

Importance

Customer insights: Businesses help businesses understand the opinion of customers about products or services.

Brand Monitoring: Tracks public spirit about a brand on platforms such as social media.

Market Research: Analyze the trend and public opinion to inform the decision making.

Crisis Management: Recognizes negative emotions quickly, allows organizations to address issues immediately.

Individualization: Customer enables corresponding marketing campaigns depending on emotions and preferences.