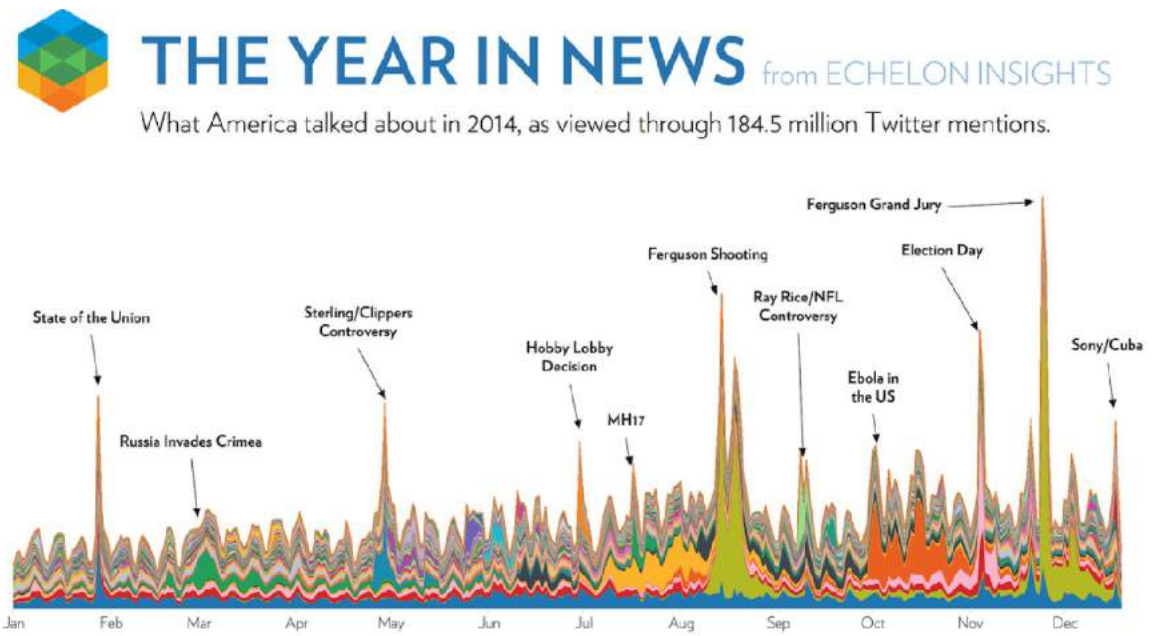
TABLE OF CONTENTS

UNIT 1: INTRODUCTION

DATA VISUALIZATION

Data visualization is the discipline of trying to understand data by placing it in a visual context so that patterns, trends and correlations that might not otherwise be detected just by looking at the data, can be presented. Its main goal is to distill large datasets into visual graphics to allow for easy understanding of complex relationships within the data. It is often used interchangeably with terms such as information graphics, statistical graphics, and information visualization. It is one of the steps in data analysis or data science. According to Vitaly Friedman (2008) the "main goal of data visualization is to communicate information clearly and effectively through graphical means. It doesn't mean that data visualization needs to look boring to be functional or extremely sophisticated to look beautiful. To convey ideas effectively, both aesthetic form and functionality need to go hand in hand, providing insights into a rather sparse and complex data set by communicating its key-aspects in a more intuitive way. Yet designers often fail to achieve a balance between form and function, creating gorgeous data visualizations which fail to serve their main purpose — to communicate information". Indeed, Fernanda Viegas and Martin M. Wattenberg suggested that an ideal visualization should not only communicate clearly, but stimulate viewer engagement and attention.

The Year in News is a good example of how expertly executed data visualization can reveal patterns and trends hiding beneath the surface of mountains of data. By analyzing 184.5 million Twitter mentions, Echelon Insights was able to provide a bird's eye view of what America was talking about in 2014.



(Source: http://echeloninsights.com/wp-content/uploads/2014/12/theyearinnews20141.png)

In his 1983 book The Visual Display of Quantitative Information, Edward Tufte defines 'graphical displays' and principles for effective graphical display in the following passage: "Excellence in statistical graphics consists of complex ideas communicated with clarity, precision and efficiency. Graphical displays should:

- Show the data
- Induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production or something else
- Avoid distorting what the data has to say
- Present many numbers in a small space
- Make large data sets coherent
- Encourage the eye to compare different pieces of data
- Reveal the data at several levels of detail, from a broad overview to the fine structure
- Serve a reasonably clear purpose: description, exploration, tabulation or decoration
- Be closely integrated with the statistical and verbal descriptions of a data set.

Author Stephen Few described eight types of quantitative messages that users may attempt to understand or communicate from a set of data and the associated graphs used to help communicate the message:

- Time-series: A single variable is captured over a period of time, such as the unemployment rate over a 10-year period. A line chart may be used to demonstrate the trend.
- Ranking: Categorical subdivisions are ranked in ascending or descending order, such as a ranking of sales performance (the measure) by sales persons (the category, with each sales person a categorical subdivision) during a single period. A bar chart may be used to show the comparison across the sales persons.
- Part-to-whole: Categorical subdivisions are measured as a ratio to the whole (i.e., a percentage out of 100%). A pie chart or bar chart can show the comparison of ratios, such as the market share represented by competitors in a market.
- Deviation: Categorical subdivisions are compared against a reference, such as a comparison of actual vs. budget expenses for several departments of a business for a given time period. A bar chart can show comparison of the actual versus the reference amount.
- Frequency distribution: Shows the number of observations of a particular variable for given interval, such as the number of years in which the stock market return is between intervals such as 0-10%, 11-20%, etc. A histogram, a type of bar chart, may be used for this analysis. A boxplot helps visualize key statistics about the distribution, such as median, quartiles, outliers, etc.
- Correlation: Comparison between observations represented by two variables (X,Y) to determine if they tend to move in the same or opposite directions. For example, plotting unemployment (X) and inflation (Y) for a sample of months. A scatter plot is typically used for this message.
- Nominal comparison: Comparing categorical subdivisions in no particular order, such as the sales volume by product code. A bar chart may be used for this comparison.

- Geographic or geospatial: Comparison of a variable across a map or layout, such as the unemployment rate by state or the number of persons on the various floors of a building. A cartogram is a typical graphic used.

Analysts reviewing a set of data may consider whether some or all of the messages and graphic types above are applicable to their task and audience. The process of trial and error to identify meaningful relationships and messages in the data is part of exploratory data analysis.

Python offers multiple great graphing libraries that come packed with lots of different features. No matter if you want to create interactive, live or highly customized plots python has an excellent library for you. In this book, we will cover the visualization processes and implementation using Python programming. To get a little overview here are a few popular plotting libraries:

- Matplotlib: low level, provides lots of freedom
- Pandas Visualization: easy to use interface, built on Matplotlib
- Seaborn: high-level interface, great default styles
- ggplot: based on R's ggplot2, uses Grammar of Graphics
- Plotly: can create interactive plots

We have dedicated a chapter on how to create basic plots using Matplotlib, Pandas visualization and Seaborn as well as how to use some specific features of each library. In another chapter we will talk about how to interpret the data. Finally we will execute a project using the techniques we learnt.

WHY DATA VISUALIZATION IS IMPORTANT

Everyday we are producing gigabytes of data. With so much data, it's become increasingly difficult to manage and make sense of it all. It would be impossible for any single person to wade through data line-by-line and see distinct patterns and make observations. Using data visualization, we can manage the data proliferation and is an important aspect of Data Science. Let's look at some of the important aspects of why data visualization is important:

Faster Decision Making

Companies who can gather and quickly act on their data will be more competitive in the marketplace because they can make informed decisions sooner than the competition. Speed is key, and data visualization aides in the understanding of vast quantities of data by applying visual representations to the data. This visualization layer typically sits on top of a data warehouse or data lake and allows users to discover and explore data in a self-service manner. Not only does this spur creativity, but it reduces the need for IT to allocate resources to continually build new models.

Let's take an example to understand it better. Let's say you are the marketing manager of your company and you need to create marketing budget for next year (2020 as on writing this book). If you've been spending a certain amount on marketing and you're happy with the results, by all means use that same formula for the future.

But what if you're not happy with your marketing results?

- Is it because your marketing budget is too small?
- Or is it because your marketing budget isn't delivering how it should?

Before increasing the amount you spend on marketing, you might want to sanity-check your marketing budget. We will be using data visualization to arrive at a decision. Let's begin to analyze how to budget for marketing and where to invest your marketing dollars.

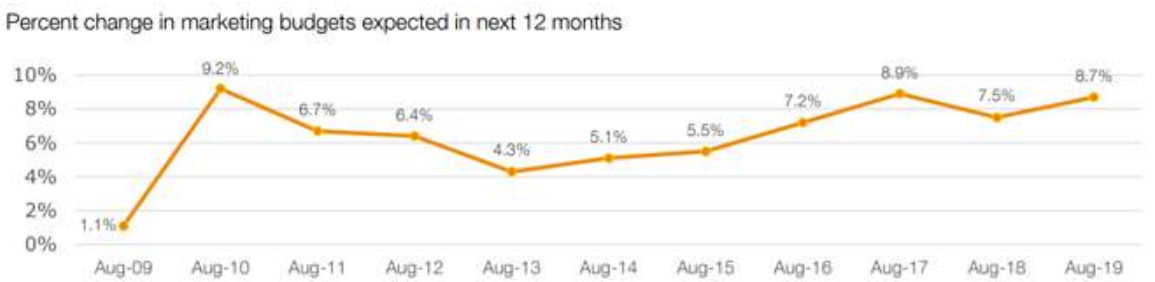
Problem Statement:

- How much are companies spending on marketing?
- Where are marketing dollars invested offline and online?
- What marketing strategies and tactics are getting results?

The answers to these questions come from a few reliable sources:

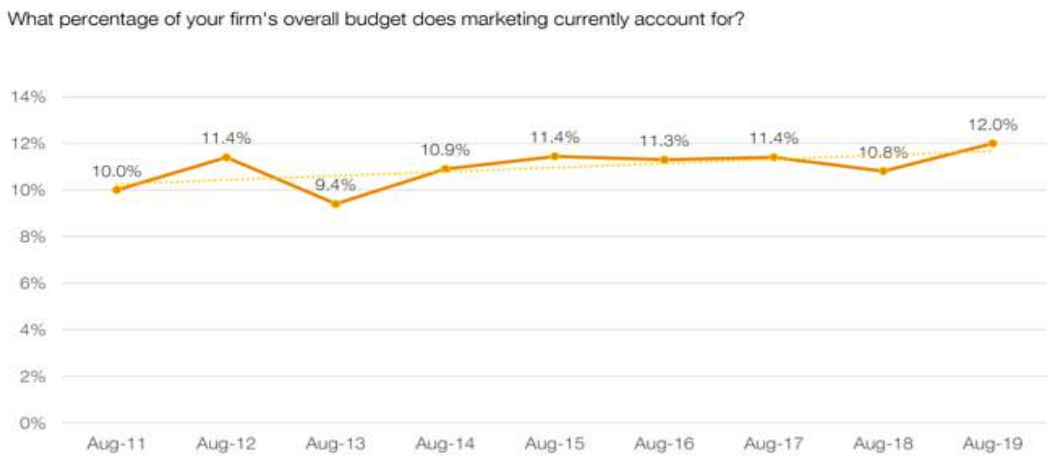
- an annual survey of Chief Marketing Officers (CMOs) from a variety of industry sectors and firm sizes
- a leading research group report about interactive (digital/online) marketing trends and predictions
- other leading digital marketing research firms

For ten consecutive years in the CMO Survey, top marketers were asked how their marketing spend was expected to change in the upcoming year. A look at how marketing budgets are changing:



At the very least, this gives a ballpark idea of how much you might want to spend. What is important to know here is the size of the companies included in the above survey and compare it to your company size. The CMO Survey figures are very unclear about whether the 'marketing budget' includes the people-related costs of the marketing team, or whether it's just money allocated to external suppliers. I tend to believe the budget includes both. When you're budgeting, include the time you spend on marketing as well as the money you spend on budgeting.

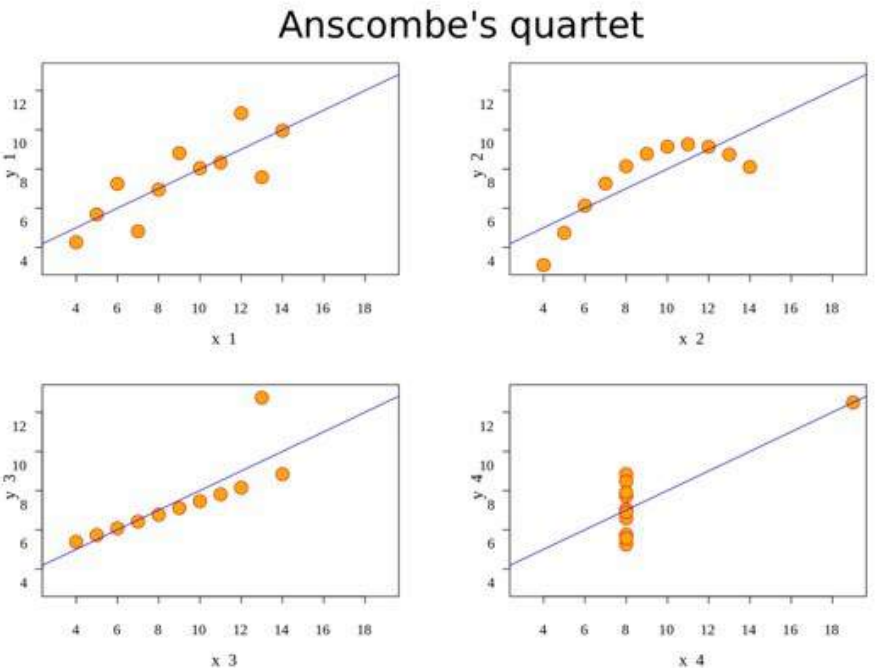
Marketing budgets as a percent of the overall firm budget has remained even more consistent, as shown in the chart below.



Consistent with the increase in marketing budgets overall, marketing budgets as a percentage of firm budgets was projected to reach 12% in 2020 according to the CMO Survey, matching the highest level they've seen.

Similar survey result can be looked to answer other questions as well. As you can see from the above example, a simple line graphs can also help in making important decisions for the companies.

Improved Insight



Data visualization can provide insight that traditional descriptive statistics cannot. A perfect example of this is Anscombe's Quartet, created by Francis Anscombe in 1973. The illustration includes four different datasets with almost identical variance, mean, correlation between X and

Y coordinates, and linear regression lines. However, the patterns are clearly different when plotted on a graph. On the next page, you can see a linear regression model would apply to graphs one and three, but a polynomial regression model would be ideal for graph two. This illustration highlights why it's important to visualize data and not just rely on descriptive statistics.

Since our sub-conscious system processes more information through vision, data visualization is a perfect solution to communicate patterns and insights from data sets. When someone sees a visualization of data, it will take less than 500 milliseconds for the eye and the brain to process what are called pre-attentive visual properties of an image. According to Colin Ware's Information Visualization: Perception for Design, he defines four pre-attentive visual properties:

- Color
- Form
- Movement
- Spatial positioning

These four components make up the composition of each data visualization and should be carefully considered for presentation.

EXAMPLES OF DATA VISUALIZATION USAGE

1. **Analysis of Long Distance Telephone bills**

Analysis of Long-Distance Telephone Bills

Following deregulation of telephone service, several new companies were created to compete in the business of providing long-distance telephone service. In almost all cases these companies competed on price since the service each offered is similar. Pricing a service or product in the face of stiff competition is very difficult. Factors to be considered include supply, demand, price elasticity, and the actions of competitors. Long-distance packages may employ per minute charges, a flat monthly rate, or some combination of the two. Determining the appropriate rate structure is facilitated by acquiring information about the behaviors of customers and in particular the size of monthly long-distance bills.

As part of a larger study, a long-distance company wanted to acquire information about the monthly bills of new subscribers in the first month after signing with the company. The company's marketing manager conducted a survey of 200 new residential subscribers wherein the first month's bills were recorded. These data are listed here. The general manager planned to present his findings to senior executives. What information can be extracted from these data?

Long-Distance Telephone Bills

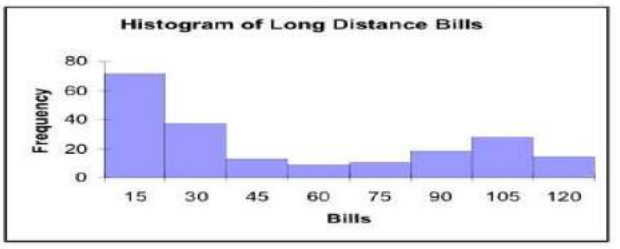
42.19	39.21	75.71	8.37	1.62	28.77	35.32	13.90	114.67	15.30
38.45	48.54	88.62	7.18	91.10	9.12	117.69	9.22	27.57	75.49
29.23	93.31	99.50	11.07	10.88	118.75	106.84	109.94	64.78	68.69
89.35	104.88	85.00	1.47	30.62	0	8.40	10.70	45.81	35.00
118.04	30.61	0	26.40	100.05	13.95	90.04	0	56.04	9.12
110.46	22.57	8.41	13.26	26.97	14.34	3.85	11.27	20.39	18.49
0	63.70	70.48	21.13	15.43	79.52	91.56	72.02	31.77	84.12
72.88	104.84	92.88	95.03	29.25	2.72	10.13	7.74	94.67	13.68
83.05	6.45	3.20	29.04	1.88	9.63	5.72	5.04	44.32	20.84
95.73	16.47	115.50	5.42	16.44	21.34	33.69	33.40	3.69	100.04
103.15	89.50	2.42	77.21	109.08	104.40	115.78	6.95	19.34	112.94
94.52	13.36	1.08	72.47	2.45	2.88	0.98	6.48	13.54	20.12
26.84	44.16	76.69	0	21.97	65.90	19.45	11.64	18.89	53.21
93.93	92.97	13.62	5.64	17.12	20.55	0	83.26	1.57	15.30
90.26	99.56	88.51	6.48	19.70	3.43	27.21	15.42	0	49.24
72.78	92.62	55.99	6.95	6.93	10.44	89.27	24.49	5.20	9.44
101.36	78.89	12.24	19.60	10.05	21.36	14.49	89.13	2.80	2.67
104.80	87.71	119.63	8.11	99.03	24.42	92.17	111.14	5.10	4.69
74.01	93.57	23.31	9.01	29.24	95.52	21.00	92.64	3.03	41.38
56.01	0	11.05	84.77	15.21	6.72	106.59	53.90	9.16	45.77

SOLUTION

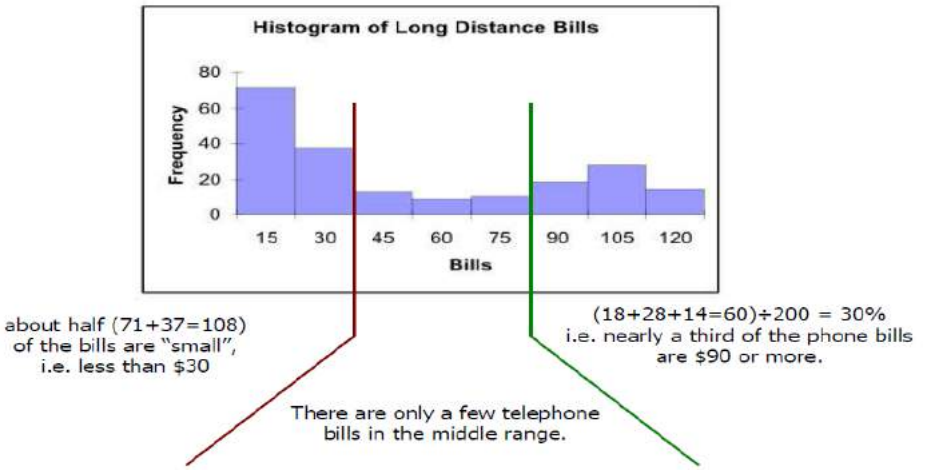
There is little information developed by casually reading through the 200 observations. The manager can probably see that most of the bills are under $100, but that is likely to be the extent of the information garnered from browsing through the data. If he examines the data more carefully, he may discover that the smallest bill is $0 and the largest is $119.63. He has now developed some information. However, his presentation to senior executives will be most unimpressive if no other information is produced. For example, someone is likely to ask how the numbers are distributed between 0 and 119.63. Are there many small bills and few large bills? What is the "typical" bill? Are the bills somewhat similar or do they vary considerably?

To help answer these questions and others like them, the marketing manager can construct a frequency distribution from which a histogram can be drawn. In the previous section a frequency distribution was created by counting the number of times each category of the nominal variable occurred. We create a frequency distribution for interval data by counting the number of observations that fall into each of a series of intervals, called **classes,** that cover the complete range of observations. We discuss how to decide the number of classes and the upper and lower limits of the intervals later. We have chosen eight classes defined in such a way that each observation falls into one and only one class. These classes are defined as follows:

Lower Limit	Upper Limit	Frequency
0	15	71
15	30	37
30	45	13
45	60	9
60	75	10
75	90	18
90	105	28
105	120	14
	Total	200



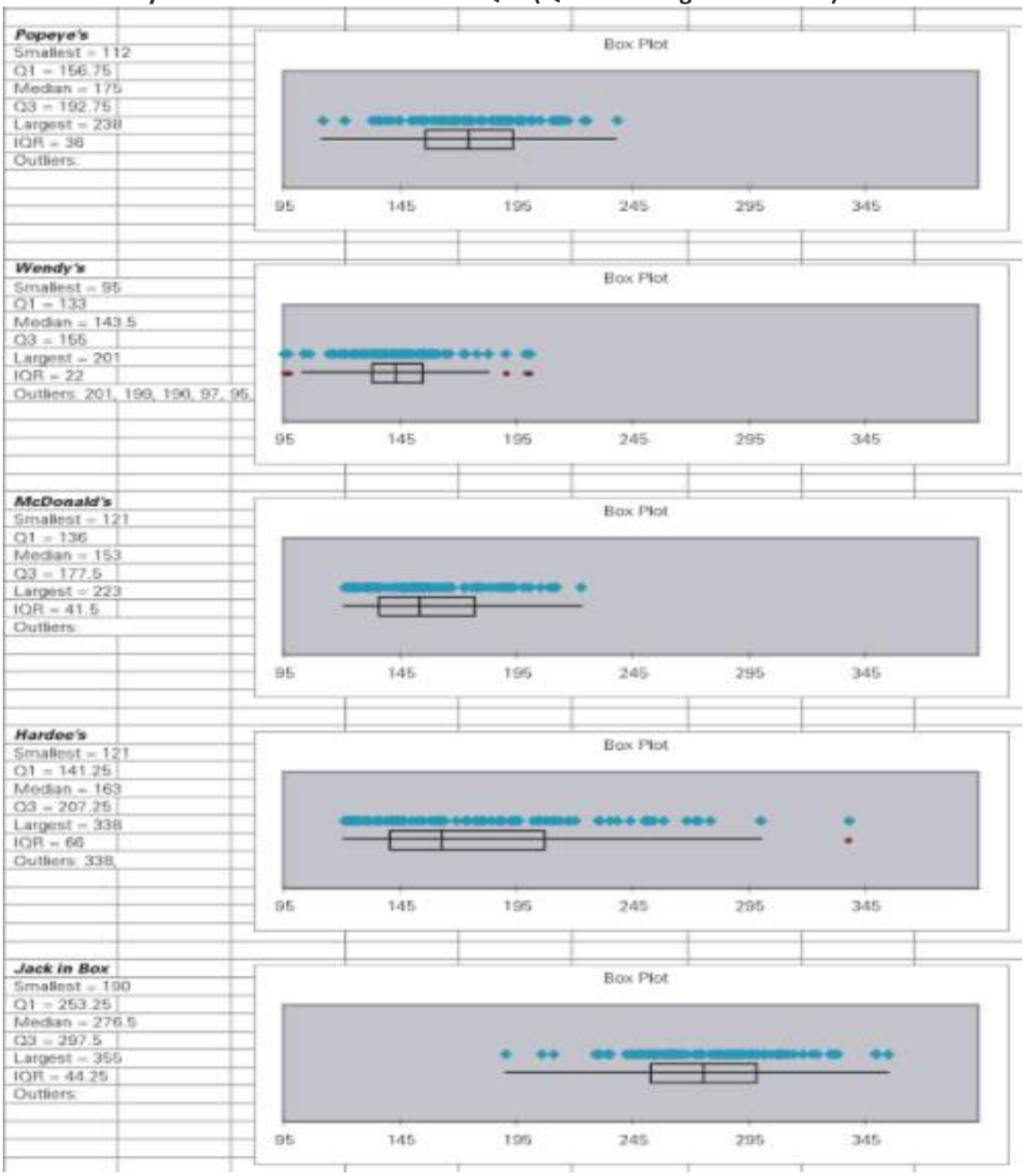
Observation



Interpretation

The histogram gives us a clear view of the way the bills are distributed. About half the monthly bills are small ($0 to $30), a few bills are in the middle range ($30 to $75), and a relatively large number of long-distance bills are at the high end of the range. It would appear from this sample of first-month long-distance bills that the company's customers are split unevenly between light and heavy users of long-distance telephone service. If the company assumes that this pattern will continue, it must address a number of pricing issues. For example, customers who incurred large monthly bills may be targets of competitors who offer flat rates for 15-minute or 30-minute calls. The company needs to know more about these customers. With the additional information, the marketing manager may suggest an alteration of its pricing.

2. Whats your observation about these QSR (Quick Serving Restaurants)?



Interpretation

Below are high level interpretation: Readers are requested to form a study group and take up the discussion about the nature of the services offered by these restaurants.

- Wendy's time appear to be the lowest and most consistent.
- Hardee's service time show more variability
- Jack in the box shows slowest time
- McDees and Hardy's are positively skewed

3. **Identify The Best Investment Options using Rental Yield Analysis**

We would like you to read the blog titled: "Identify The Best Investment Options using Rental Yield Analysis by Santosh Krishna on housing.com
https://housing.com/news/how-rental-yield-analyses-can-help-you-invest-wisely-in-bangalore-mumbai/

You will see the author is talking about the housing prices in Mumbai versus Bangalore. He uses few graphs to put across his point, but nowhere he is focusing more than required focus on the plots. His focus is on the analysis and plots he uses to support his points. That's how our analysis has to be – use visualization to support your point.
You will come across number of such articles on internet. My idea here was to introduce to the topic and convey to you the role and importance of visualization.

DATA VISUALIZATION PROCESS

In this section, we will talk about the process to perform data visualization and rest of the chapters we will see how to implement them. Here are the 7 processes one has to follow to perform data analysis techniques efficiently:

1. Develop your research question

It is important to have a clear understanding of the goal of your research. This will determine what sort of data is needed, the type of analysis necessary, and the types of visualizations that would be most effective to communicate your explorations or findings.

2. Get or create your data

The Library provides:

- access to a large collection of numerical, statistical and geospatial data. There is also a great wealth of open data freely available for download on the web.
- advice and technical assistance with the design, creation, and dissemination of surveys using the Qualtrics web survey platform to assist you in collecting your own data.
- assistance in creating your own data through methods such as digitizing imagery and documents, collecting data from APIs, and scraping data from web sources.

3. Clean your data

This is an essential step to perform before creating visualization. Clean, consistent data will be much easier to visualize. Clean data is data that is free of errors or anomalies which may make it hard to use or analyze the data. Starting from a clean dataset allows you to focus on creating an effective visualization rather than trying to diagnose and and fix issues while creating visualizations.
Data cleaning tasks will be very dependent on the dataset that you're working with. In most cases, data cleaning involves:

- Removing unnecessary variables
- Deleting duplicate rows/observations
- Addressing outliers or invalid data

- Dealing with missing values
- Standardizing or categorizing values
- Correcting typographical errors

4. Choose a chart type

It is important to pick a chart or graph which will best communicate the message you wish to tell your audience. The first step to choosing a chart is to determine what message you're trying to deliver. Are you:

- Showing how variables compare to each other?
- Showing relationships between variables?
- Showing patterns in the data?
- Showing how the whole dataset can be broken down into smaller parts?

Picking one of these can help you to select the chart that will work best for you. In this book, we will be covering number of resources to help you to choose the right one for your work.

5. Choose your tool

There are many tools to choose from ranging from freely available open web based tools to licensed desktop tools. In this book, our focus will be using various features of Python but we will briefly talk about other tools that are available to use apart from Python.

6. Prepare data

The data preparation steps you'll need to go through are determined by the type of chart or visualization you'd like to create and the tool which you have chosen. Once you've selected your chart you may need to perform transformations to create the required data.

Typical data preparation tasks include:

- Formatting columns appropriately (numbers are treated as numbers, dates as dates)
- Convert values into appropriate units
- Filter your data to focus on the specific data that interests you.
- Group data and create aggregate values for groups (Counts, Min, Max, Mean, Median, Mode)
- Extract values from complex columns
- Combine variables to create new columns

7. Create chart

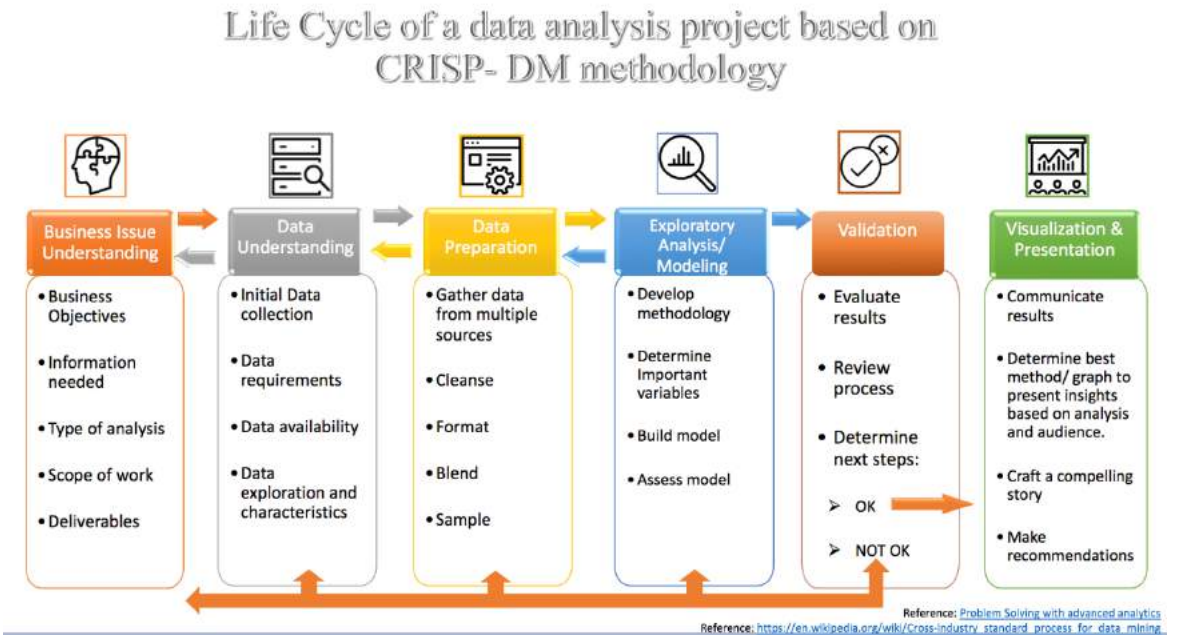
Following the procedures specific to your tool, the general process for creating charts includes:

- Import data into the software
- Select the chart type you wish to create
- Evaluate the effectiveness of the chart.
- Refine by applying design principles. The way in which you design your chart can have a big impact on the effectiveness of the chart. Consider these design principles.

While designing a data analytics project, we are often left wondering where to begin with in the first place? From data collection, cleaning, exploration, analysis and visualization, there is a lot

that needs to be done in order to derive an insight that is - actionable & profitable, for the business.

There seems to be a no set way to approach this problem. However, in order to provide a framework to organize the work needed by an organization and deliver clear insights from data, it's useful to think of it as a cycle with different stages. A sample framework is depicted in a figure on the next page. In this figure we depict a data science framework, breaking it down to each step that are important for solving a business problem with data visualization. In the chapters to come, we will get ourselves familiarized with the whole process in a simpler way.



Now, let's get ready to deep dive into each of the topics and learn more about them and at the sametime be ready to dirty our hands with Python code!

UNIT 2: DEVELOP YOUR RESEARCH QUESTION

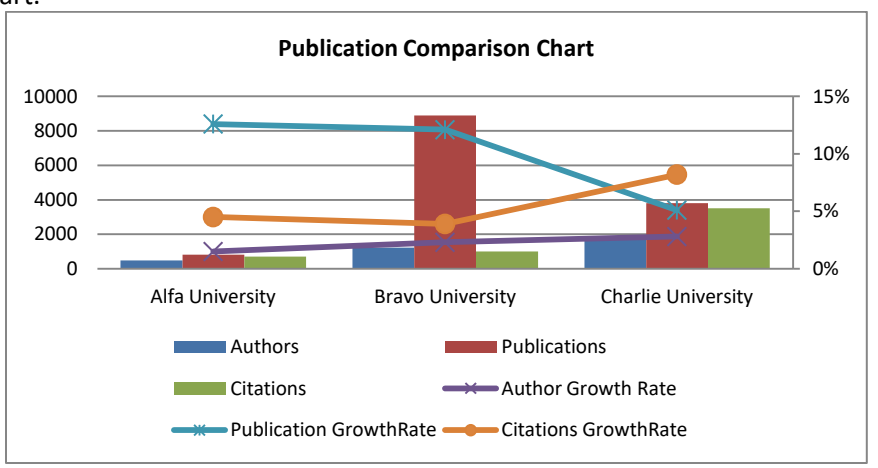
When creating a visualization, the first step is to be clear on the question to be answered – or alternatively answering the question: "What business problem am I solving?" or to put it in the simple word "How will the visualization help the reader?"

The business need defines the problem that we are seeking a solution for. At the start of the project, the focus is to get a clear understanding of the overall scope of the work, business objectives, information the stakeholders are seeking, the type of analysis they want you to use, and the key deliverables. Defining these elements prior to beginning the analysis is important, as it helps in delivering better insights. Also, it is important to get a clarity at the beginning as there may not be another opportunity to ask questions before the completion of the project.

Having a clear question to answer helps avoid a common problem in data visualization: Comparing "apples" to "oranges." Consider a hypothetical dataset (see Table below) in which we have information on an institution's total number of authors, publications, citations and their respective growth rate for a given year.

	Authors	Publications	Citations	Author Growth Rate	Publication GrowthRate	Citations GrowthRate
Alfa University	490	810	710	1.50%	12.60%	4.50%
Bravo University	1220	8900	1010	2.30%	12.10%	3.90%
Charlie University	1900	3800	3500	2.80%	5.10%	8.20%

A bad example of visualization is shown in Figure below, where all the variables are included in the same chart.



Plotting variables of different types in the same chart is seldom a good idea because a distracted reader may be misled to compare variables that are not comparable. For example, it does not make sense to observe that all the institutions have fewer authors than the total number of publications, or that the publication growth rate rises from Alfa University, to Bravo University to Charlie Institution. Busier graphs are just harder to read and process, and this is the case when

you have multiple y-axes; it is not always clear which variable corresponds to which axis. Simply put, a bad visualization confuses rather than clarifies.

DEVELOPING A BUSINESS QUESTION

An objective statement is an explanation of company goals, including what the company would like to achieve and the overall ideals of the organization. Objective statements can vary in length and detail depending on the complexity of the company's goals. Organizations can set objectives to improve customer service, enhance worker productivity and increase profits. Each objective statement should include information about how the company plans to achieve business goals.

Goals
Objective statements should ultimately identify company goals. They should also include a statement of purpose and an explanation of how the company plans to achieve the stated goals. For example, if the goal of an organization is to improve customer retention, the objective statement could read "to improve customer retention 20% by implementing a customer loyalty program and extending customer service hours." This objective statement includes the organization's goal as well as what steps the company will take to achieve the goal.

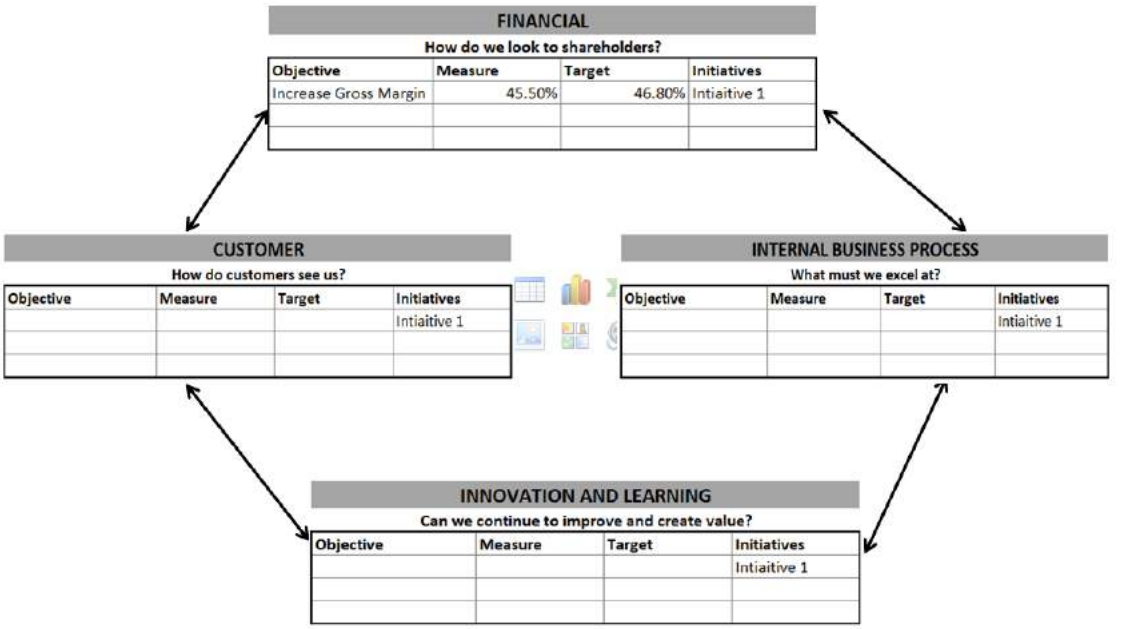
Business Strategy
An objective statement should include a business strategy. A business strategy is a specific plan of action to accomplish a goal. The strategy should describe how the company intends to meet the objective. For example, if the company would like to expand a product line, the objective should identify various strategies to make that happen, such as "Spend time each month identifying new products" or "Attend new product presentations from current suppliers."

Measurable Information
Company objectives should be measurable. Therefore, objective statements should include how the company will measure the outcome of the goal. This is typically achieved by including a statement about where the company is currently and where the company would like to be. For instance, if the company would like to decrease employee turnover, the statement would include current retention rates as well as desired retention rates. For example, this part of the statement could read "*To decrease employee turnover from 20% to 12% within the next 12 months*." By including quantifiable information, the organization can effectively measure the objective's outcome.

Skills and Strengths
An objective statement should indicate the skills, strengths and areas of expertise of the company. For instance, if the organization currently holds 50 percent market share and a trusted brand, the objective statement should include information about these strengths. An example of this would be "to build upon our brand image to improve customer relations." The objective statement should also include information regarding the organization's reputation if it is known for providing exemplary customer service.

Methodology

One of the most popular methodologies is to use The Balanced Scorecard. The Balanced Scorecard is a management system. It's a way of looking at your organization that focuses on your big-picture strategic goals. It also helps you choose the right things to measure so that you can reach those goals. The balanced scorecard allows managers to look at the business from four important perspectives.



It provides answers to four basic questions:

- How do customers see us? (customer perspective)
- What must we excel at? (internal perspective)
- Can we continue to improve and create value? (innovation and learning perspective)
- How do we look to shareholders? (financial perspective)

While giving senior managers information from four different perspectives, the balanced scorecard minimizes information overload by limiting the number of measures used. Companies rarely suffer from having too few measures. More commonly, they keep adding new measures whenever an employee or a consultant makes a worthwhile suggestion.

Traditionally, companies have judged their health by how much money they make. Financial measures are definitely important, but they only give you part of the picture. They focus on the short-term, and you're trying to build an organization to stand the test of time. The name "balanced scorecard" comes from the idea of looking at strategic measures in addition to traditional financial measures to get a more "balanced" view of performance. It's this focus on both high-level strategy and low-level measures that sets the balanced scorecard apart from other performance management methodologies.

In the next section, we will look at some examples of creating your problem statement.

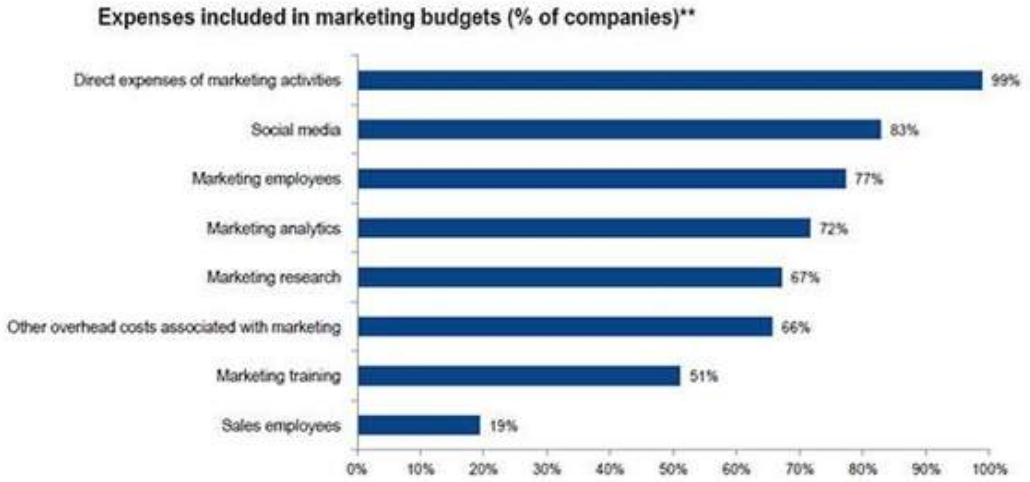
EXAMPLE: BUSINESS CHALLENGES FOR A MARKETING MANAGER

Marketing Management

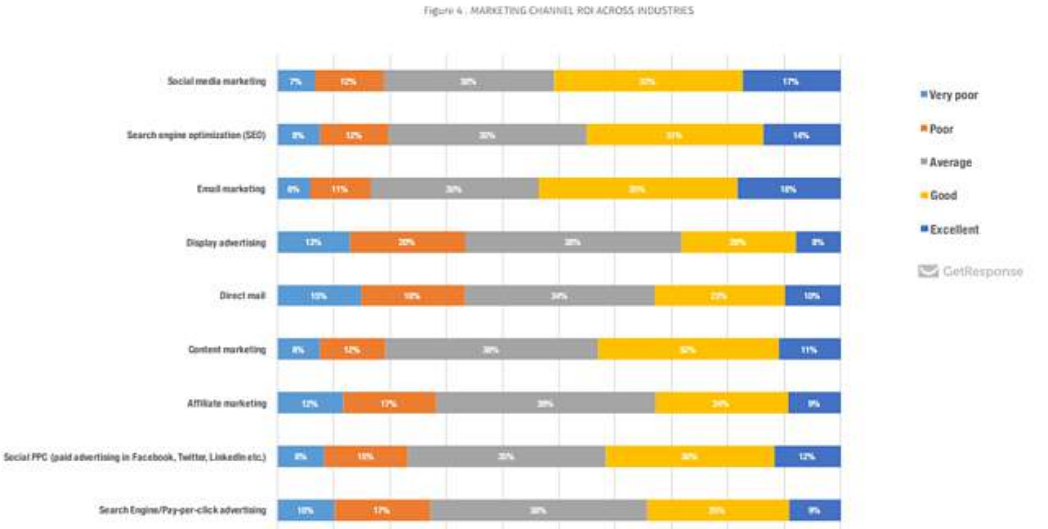
We have discussed about this in our first chapter. Let's look at few more business questions which would be of interest to a marketing head. In this section, we will look at the business objective and present a graph that would help in making an informed decision.

***Source**: Annual survey of Chief Marketing Officers (CMOs) from a variety of industry sectors and firm sizes by Webstrategies.com*

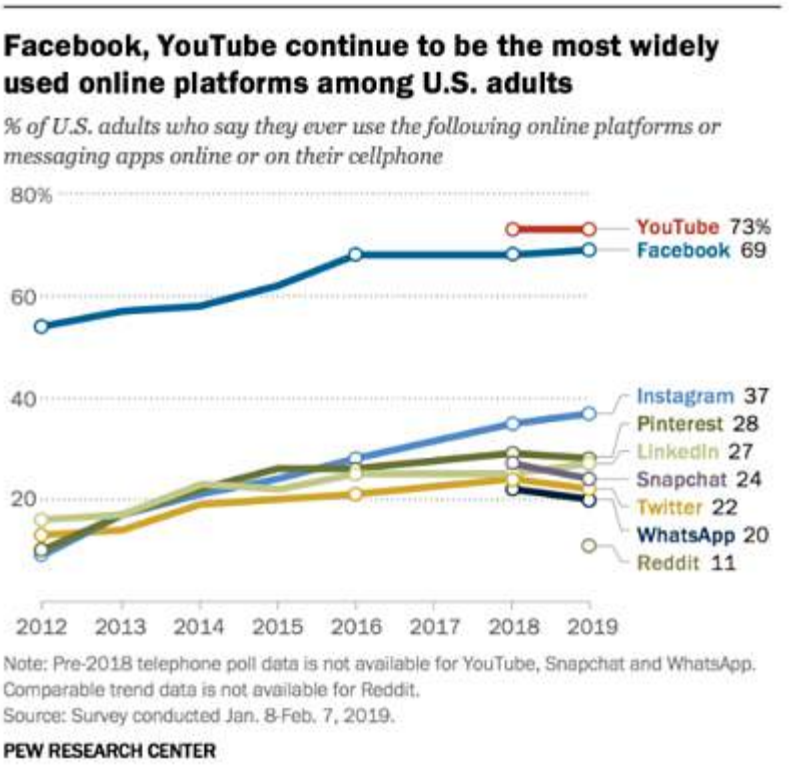
- How to allocate marketing budgets across channels?



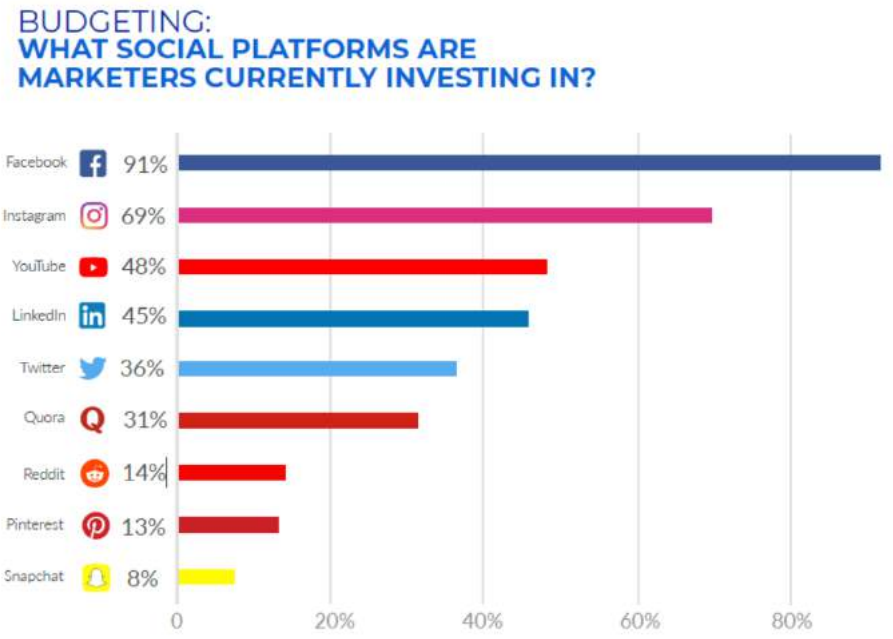
- Which marketing strategies & tactics are getting the best results?



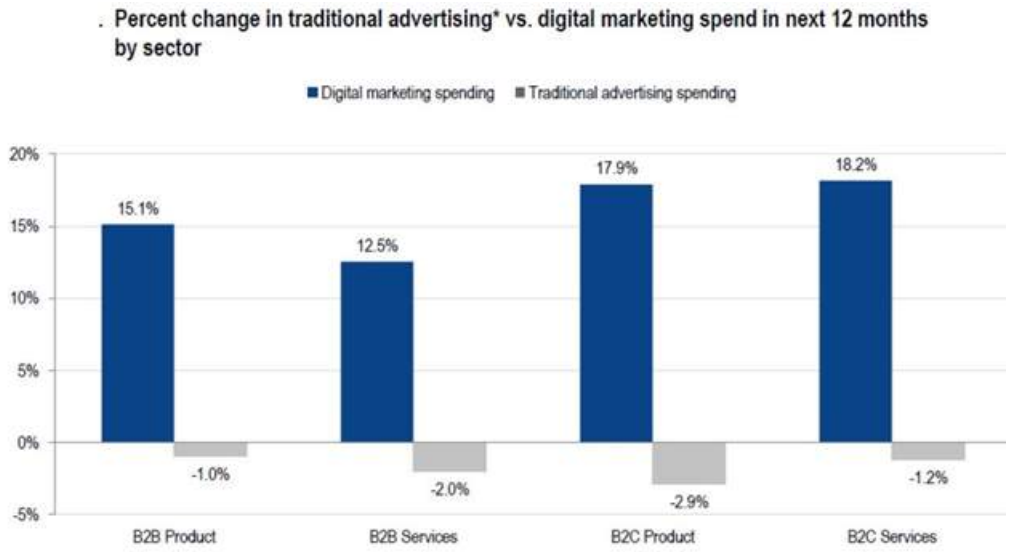
- Changes in traditional versus digital marketing spend and whats the impact?



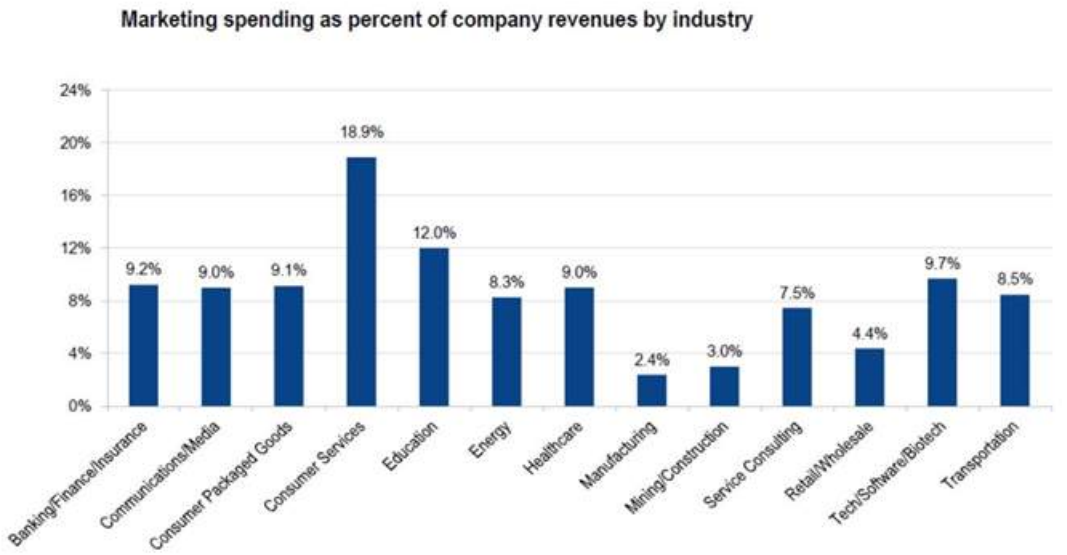
- What percentage of marketing budget spent on digital?



- How much should your firm budget for marketing?



- What is the right strategy for my business?



EXAMPLE: REALIZING BENEFITS FROM RETAIL ANALYTICS

One major issue regarding analytics is that retailers may feel as if there are too many analytical options to explore. The breadth of analytical processes and applications suggests that retailers need to selectively target investments in analytics based on their strategies and industry positions. In a difficult economy, retailers may also need to adopt first those analytical

applications that have saved money for other adopters, including pricing and assortment optimization. There is little doubt, however, that the aggressive adoption and exploitation of analytics has led to competitive advantage among some of the world's most successful retailers.

Retail Area 1: Assortment Optimization and Shelf Space Allocation
Business problem to solve: Using analytics to determine what products to offer in what quantities

Managerial Implications: Assortment and shelf space optimization impacts one of the broadest sets of business processes among the categories available to retailers. Therefore, retail executives should have a plan and sequence in mind for how they will implement such a broad capability. For example, assortment and space optimization are also closely linked with price optimization, in that it is difficult to optimize prices on the wrong product inventory. Assortment plans must be developed with the constraints of replenishment and packing that are imposed along with an awareness of available shelf space.

Examples: Lowes, the US home improvement retailer, manages over 800 planograms per store in collaboration with over 150 key suppliers. Retailer Marks & Spencer achieved $2.5 million in labor efficiencies and $1.5 million in operating improvements, as well as a number of intangible improvements, through planogram automation and optimization.

Retail Area 2: Customer-Driven Marketing
Business Problem to Solve: Use of customer data to segment, target, and personalize offerings

Managerial Implications: Analytics are not the most difficult issue in customer-driven marketing for offline retailers; getting and maintaining unique customer information is often more problematic. Most retailers grasp the importance of a unique means of identifying customers, so they have already considered or adopted loyalty programs. Firms relying on credit card-based identification can typically uniquely identify no more than 50% of customers, and data protection laws have placed additional restrictions on using credit card purchase data to uniquely identify customers.

Examples: Tesco has profited greatly from the introduction of its loyalty card program ClubCard. 80% of Tesco's sales can be tracked through ClubCard. The retailer provides rebates of 1% of customer purchases—historically by direct mail, but increasingly by email. The online discount retailer Overstock.com uses targeted and event-driven email marketing to customize category and product offers and even pricing under different circumstances. Targeting email offers has led to a 50% revenue lift from the email channel, and average order size increased by 6%.

Retail Area 3: Fraud Detection and Prevention
Business Problem to Solve: The use of analytics to detect and prevent online and offline fraud

Managerial Implications: Because fraud typically involves employee theft, payment mechanisms, or customer returns, retailers should put systems and processes in place that address all three

issues. Payment-related fraud should be addressed in collaboration with banks and providers of private-label credit cards. Customer-oriented fraud often takes place in returns, so retailers should address both clearly fraudulent returns and return policies that facilitate fraud. In most cases, analytical systems and processes do not identify fraud sufficiently clearly to enable prosecution, so employees must be trained to pursue the highest-likelihood cases of potential fraud with further investigation.

Examples: Online retailer Amazon.com has an aggressive program to identify and prevent credit card fraud, which in its first six months led to 50% reductions in fraud. Amazon uses a scoring approach to identify the most likely fraud situations in customer purchases. Canadian department store retailer Hudson's Bay Company (HBC) has used analytics to fight returns fraud—when a customer brings back a shoplifted item or reuses a sales receipt to return an item multiple times.

Retail Area 4: Integrated Forecasting
Business Problem to Solve: The use of statistical forecasting to support multiple functions

Managerial Implications: Implementing integrated forecasting is not only a technological initiative, but also a change in business processes and organizational roles. Most organizations have to reassign forecasting to a centralized organization to ensure that integrated forecasts are produced and employed throughout the organization.

Examples: Office products retailer Staples has adopted a new approach to sales forecasting that was originally used for making real estate decisions on over 5,000 potential store sites. It takes eight million data transactions each week as input, and forecasts weekly and daily sales for more than 1,100 US stores. Waitrose has developed a new system for store-level sales and demand forecasting. It takes into account holidays (including Pancake Day), promotions, and seasonality for predicting demand and feeding replenishment processes.

Retail Area 5: Localization and Clustering
Business Problem to Solve: Tailoring multiple aspects of retailing to local stores or similar clusters

Managerial Implications: Localization to some degree is almost always desirable, but approaches based on store clusters may be just as effective as those based on individual stores, and are much easier and more economical to implement.

Examples: WalMart has created a "store of the community" localization program that tailors store formats, assortments, shelf space allocations, and department layouts by cluster. Macy's has embarked upon a major localization initiative called "My Macy's." The primary focus is on localized allocations by item, color, and size.

Retail Area 6: Pricing Optimization

Business Problem to Solve: Using analytics to determine the optimal pricing of products and services through their lifecycles

Managerial Implications: While pricing optimization usually results in margin improvements and pays back its investment quickly, it does normally require a substantial upfront investment. This has been the primary obstacle for smaller retailers thus far leading to availability of Software as a Service (SaaS) options with far lower initial capital investment requirements. Pricing optimization should ideally be undertaken along with other approaches to merchandising optimization such as shelf space and assortment management.

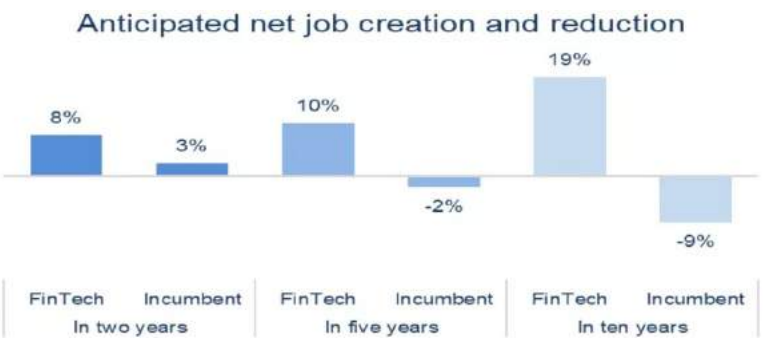
Examples: Drugstore retailer Duane Reade increased unit sales of baby formula by 14% and of disposable diapers by 10% in a 20-store trial, and then expanded the use of price optimization in all its stores for non-pharmaceutical products. Canadian apparel retailer Northern Group Retail reports that price optimization software has helped it increase gross margins by 4.5% (a 2% gain the first year and an additional 2.5% in the second).

Other Business problem of objectives areas in Retail includes:

1. Product Recommendation: Using analytical approaches to recommend product offerings for particular customers
2. Real Estate Optimization: Using analytics to optimize real estate sites and formats
3. Supply Chain Analytics: Optimizing inventory, replenishment, and transportation costs with analytics
4. Workforce Analytics: Optimization of staffing with regard to cost, customer shopping patterns, and locations
5. Store-Level Empowerment: Giving store managers and employees the ability to analyze their businesses

OTHER EXAMPLES OF PROBLEM STATEMENT

Financial Services



- To identify important insights in data and prevent fraud. Data mining can also identify clients with high-risk profiles or use cyber surveillance to pinpoint warning signs of fraud.
- The insights can identify investment opportunities, or help investors know when to trade.

- Sentiment/News Analysis: Hedge funds hold their cards tight to their chest, and we can expect to hear very little by way of how sentiment analysis is being used specifically. However, it is supposed that much of the future applications of machine learning will be in understanding social media, news trends, and other data sources, not just stock prices and trades.

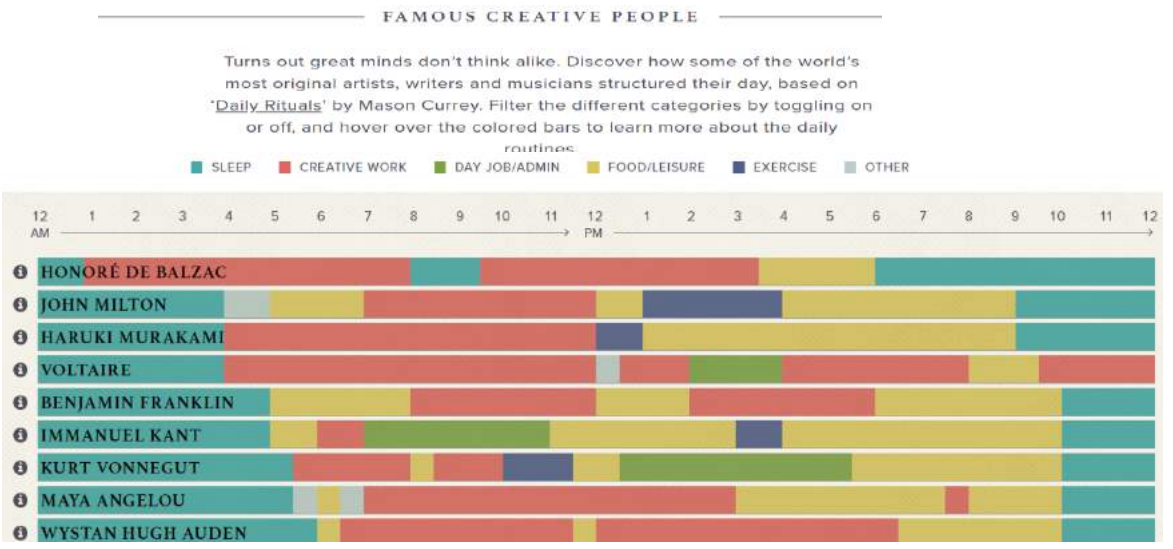
Healthcare

- Radiology - AI solutions are being developed to automate image analysis and diagnosis. This can help highlight areas of interest on a scan to a radiologist, to drive efficiency and reduce human error.
- Drug Discovery - AI solutions are being developed to identify new potential therapies from vast databases of information on existing medicines, which could be redesigned to target critical threats such as the Ebola virus.
- Patient Risk Identification - By analysing vast amounts of historic patient data, AI solutions can provide real-time support to clinicians to help identify at risk patients.
- Primary Care/Triage - Multiple organisations are working on direct to patient solutions to triage and give advice via a voice or chat-based interaction.

PURPOSE OF DATA VISUALIZATION

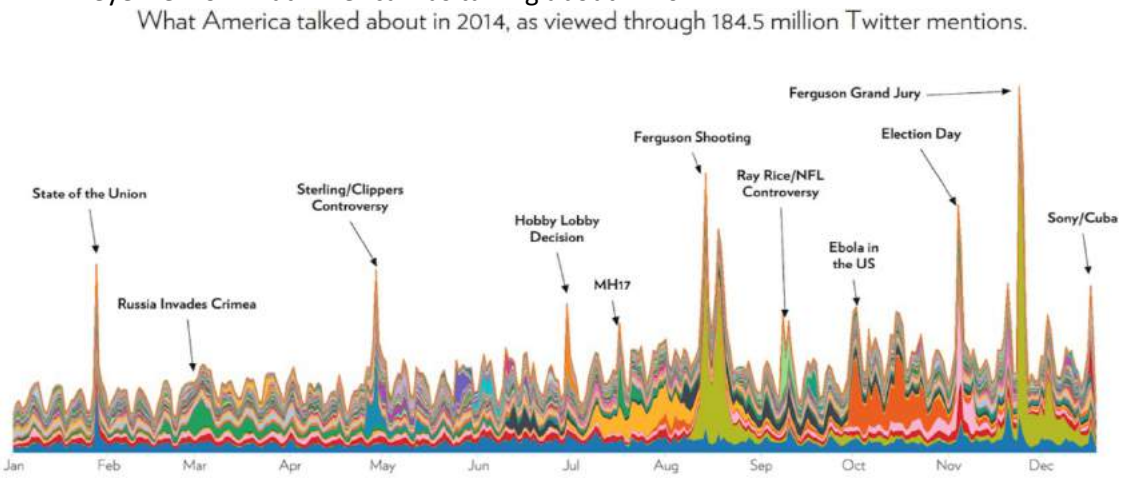
Your Data visualization output has to be:

1. **Interactive**: It should combine all the necessary ingredients of an effective and engaging piece: reams of data into a single page; uses color to easily distinguish trends; allows the viewer to get a global sense of the data; it engages users by allowing them to interact with the piece; and it is surprisingly simple to understand in a single glance.



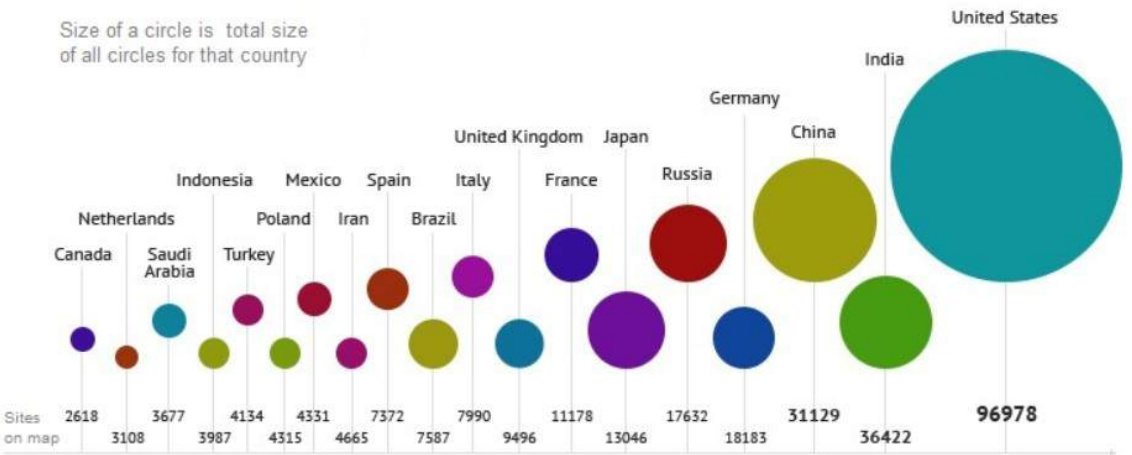
Source: https://podio.com/site/creative-routines

2. **Reveals trends:** Below is a good example of how an expertly executed data visualization can reveal patterns and trends hiding beneath the surface of mountains of data. By analyzing 184.5 million Twitter mentions, Echelon Insights was able to provide a bird's eye view of what America was talking about in 2014.



3. **Use Animation and multiple colors Only If required**. Interactive charts are good way to represent data and there are tools available which will help in creating such charts.

4. **Use symbols and metaphors**: An effective way to communicate complex ideas is by using symbols and metaphors. Take, for example, this ambitious data visualization of the Internet created by Ruslan Enikeev. By using the metaphor of planets in a solar system, Enikeev is able to create a "Map of the Internet" that helps users visualize the relative reach and influence of every site out there. The amount of website traffic, for example, is represented by the size of the circle on the map.



5. **Put data into context**: One of the great strengths of data visualizations is their unsurpassed ability to put isolated pieces of information into a bigger context so that your audience has no problem in understanding.

6. **Saves time and effort:** An effective data visualization has ability to summarize a ton of information and, in the process, save you time and effort.
7. It **explains a process**
8. It **stimulates the user's imagination**
9. It presents **data beautifully and in an easy manner to understand**
10. It **tells a story**
11. It **empowers the user to take a decision**
12. It is **educational and informative**

MINI PROJECT: PERFORMING ANALYSIS

Background

Every modern-day technology company strives to increase its revenue by tapping into its existing customer base. There are several ways of doing this - sale of service contracts or spare parts or accessories, or upgrades to newer versions of the same product or sale of a newer better product. In any of these cases, it is essential for these companies to have a comprehensive information about the customer and the product being used by that customer in order to make the best sales/service propositions. In this work, we will be taking an example of one of world'd well-know consumer products brand and we will call it Company C for our reference. We will be using some real data along with some hypothetical data. Our subject Company C is one such organization that collects and maintains "install base data"– a collective term comprising of customer and product information – as an input for its services marketing.

Problem Statement

As with any other commercial organization, Company C strives to maximize its revenues with new service offerings and to minimize its operating costs while delivering service. As a first step, Company C tries to leverage on its install base data and derive insights to optimize its current service delivery processes. The next level of deriving insights into the install base data would be for the purpose of approaching existing customers with newer and better service propositions.

The outcomes of such analyses is particularly interesting for the Service Marketing group at the Company C business unit which develops new service strategies that are taken up for implementation by the customer-facing marketing teams. The traditional way of doing such analyses would be to download all the relevant data into spreadsheet tools like Microsoft Excel and create table summaries and visualizations that are built in. But this poses two challenges that seriously limit such an analysis. Firstly that install base data is huge, with thousands of records and secondly, that the visual analytics capabilities provided by spreadsheet tools is very limited in terms of the interactions that can be performed for exploring the data. Hence there is a clear need for a custom visualization dashboard, which brings us to our research question:

- How to design a visualization tool that helps to analyze install base data for formulating better service propositions to existing customers?

In order to make better service propositions to existing customers, the data analyst needs to have access to two kinds of information.

One, an overview of the entire install base in order to decide which existing customers should be approached.

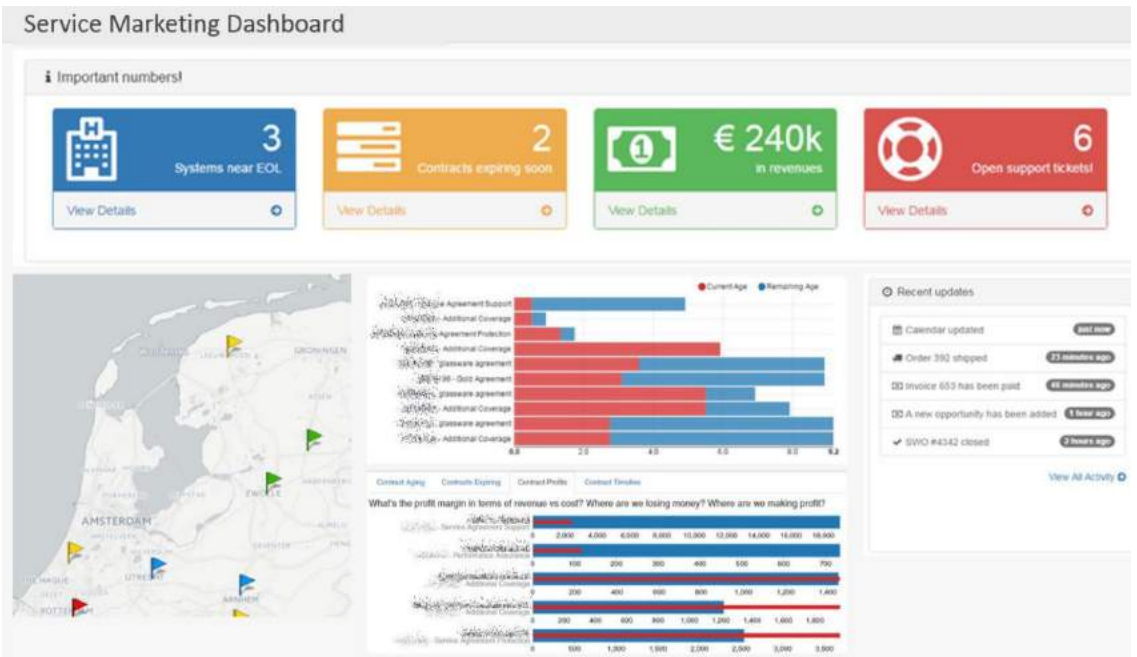
And two, an overview of the service history of each installed product that the customer is using.

Thus, the goal of this analysis is to design and implement a visualization based on specific business requirements using a standard methodology. This goal is split into two objectives: to develop a visualization that provides an overview of install base and to develop a visualization that summarizes the service activities performed on an individual system.

Result

The initial version of the overview of install base took into consideration the domain expert's interest in checking if it would be possible to encode more colors onto the icons than just based on profitability. An example of such a case is when there is an opportunity to sell a new upgrade to a customer using an old version of the installed product. The early prototypes developed did demonstrate that this was possible, but since the requirements was not clearly elicited and it was unclear whether all the data required to show such information was available, this idea was parked for a future implementation. Figure shown on the next page, shows the screenshot of a dashboard view that was developed in the first version of the prototype in order to provide an overview of the install base.

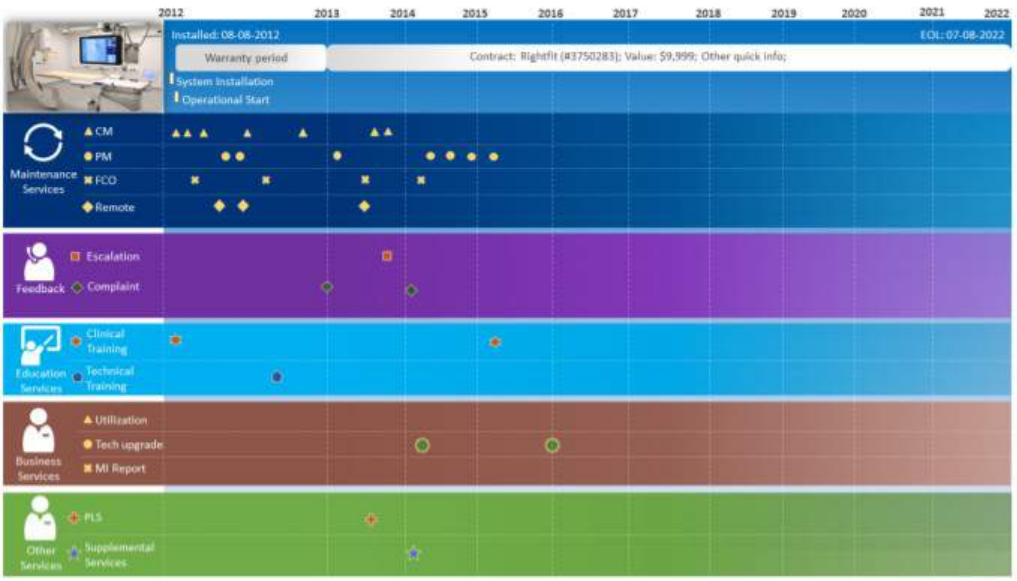
Due to the sensitive nature of the data, the graphs are purposely made not readable. But this will give you an idea how dashboards are designed and what kind of information might be useful for the audience.



As can be observed in the figure above, the screen consists of two rows – an infographic row that displays items of interest using large icons on the top and a bottom row with geographical map accompanied by two other visualizations that serve as filters. The bottom row is split into three columns in which the map is on the left. The center column is contains visualizations that allow the domain expert to filter on either the system age or the contract profitability. A point to note, this prototype did not yet have the color coding for contract profitability on the filter visualization on the bottom center. The third column on the right was to give the user an overview of the recent activities that were a part of service delivery.

A quick feedback session with the domain expert resulted in the opinion that the map is preferred to be full screen to provide a larger view of the geographical area and that the filters be displayed using a floating menu area that could be collapsed into a button or a bar on one of the sides.

Service touchpoints: a static visualization



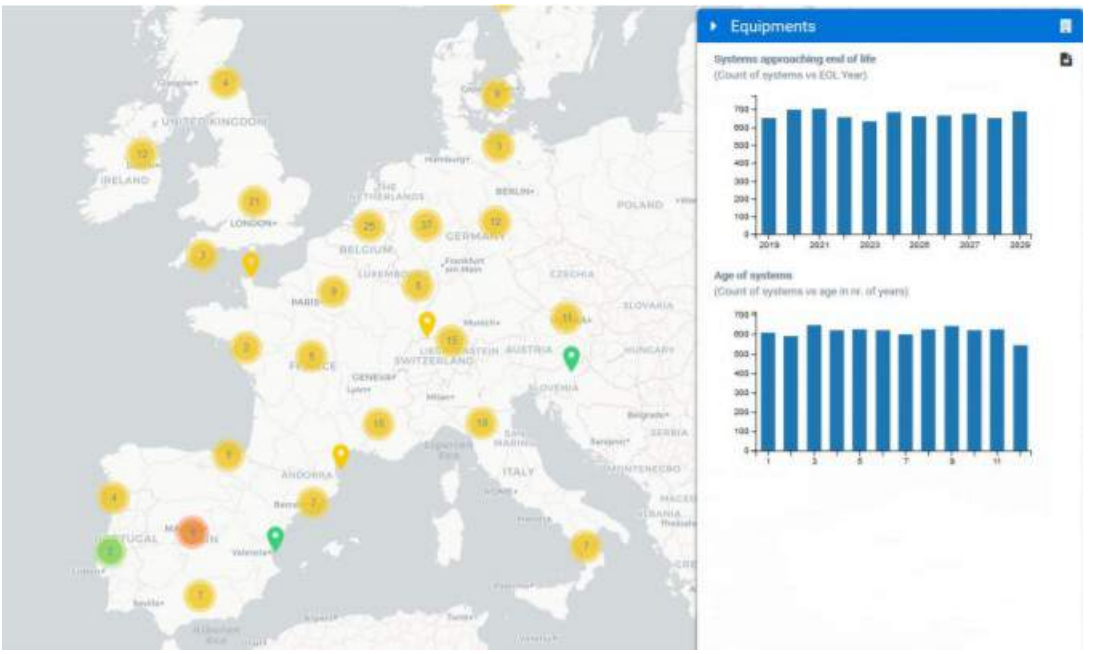
A prototype for visualizing service touchpoints was developed as shown in Figure above. The top most row displayed the time period of the lifetime of the installed product along with the period of the contracts encoded as bars. The five different categories of service touchpoints are displayed as five swim-lanes of different colors in which each type of service touch point gets its own row. In each row, the exact occurrence of the corresponding event is marked with a particular shape in a color that contrasts the background.
The horizontal x axis is a time scale spanning the lifetime of an installed product while the vertical y axis is a named scaled with of each values on the scale representing a specific service activity.

After trying out the prototype on a few installed products in the data set, it was discovered that the data was consistent and available only in case of touchpoints that were recorded as a service

work order, for example, a corrective maintenance (CM) activity or a planned maintenance (PM) activity. Other data such as training or utilization had data spread in different data sources and posed a big challenge to gather such data and in turn difficulties in evaluating the visualization if implemented. After discussion with the domain expert, it was decided to park this approach for a future implementation and concentrate only on the financials of service work order data that was available and was possible to evaluate.

Overview of Installed Base

After another iteration of prototype development and feedback by the domain expert, a working application as per the needs stated was developed.



The application opens up with a high level view of the installed base that depicts the geographical spread of the installed products using location markers and graduated symbols. The symbols are circles with a number on a color coded background of red, green or yellow colors. The color coding has its own meaning. Each marker represents the geographical location of an installed product and each symbol represents a set of installed products in and around the region in which the symbol is present. The number within the symbol depicts the exact number of markers around the region of the symbol.

Conclusion

This project started with the goal of researching solutions to the business problem faced at the services marketing group of Company C. The need for an information visualization for analyzing the existing install base was recognized and a research question, “How to visualize install base

data in order to formulate better service propositions to existing customers?" was formulated project.

The research question led to the defining of the goal for the project based on which a field study was conducted to discover any existing visualization tools that can already be used instead of developing a new one.

The design study began by setting up possible collaborations with key people from the business. The wide set of collaborators was narrowed down and their roles were set early in the project. Then the detailed requirements were discovered and gathered using the use-case documentation methodology borrowed from software requirements engineering.

In the following design stage, abstractions and encodings were developed in line with the use cases. Quick prototypes were implemented in order to validate the design decisions and updates to the design were made as per feedback received along the process. The visualization was completed by setting one use case – visualizing service touchpoints – out of scope and by leaving it open for future development. The final implementation was evaluated using various methods as directed by the seven guiding scenarios of information visualization. The results of the evaluation were discussed in detail concluding that the domain expert who evaluated the visualization was satisfied with the tool.

Source: Install base visualization for service marketing by Kumar, K.B. (2018)

This always good to get access to real work down and learn from them. Now let's move to next chapter where we will learn to get access for the data.

UNIT 3: GET OR CREATE YOUR DATA

We always have to fight for the right and useful datasets to solve the business problems presented to our team. It is important to take right dataset for data visualization or that matter machine learning projects. In this chapter we will talk about the importance of finding right dataset and how to get them for you to practice various concepts. Our intention is when you move to the real world projects, you take the learning from these dataset and will help you to appreciate the importance of data.

WHAT IS A DATASET

To understand what a dataset is, we must first discuss the components of a dataset. A single row of data is called an instance. Datasets are a collection of instances that all share a common attribute. To solve a business problem, you will be using more than one datasets, each used to fulfill various roles in the system. Data can come in many forms, but machine learning models rely on four primary data types. These include numerical data, categorical data, time series data, and text data.

Numerical data (or Quantitative data)

Numerical data, or quantitative data, is any form of measurable data such as your height, weight, or the cost of your phone bill. You can determine if a set of data is numerical by attempting to average out the numbers or sort them in ascending or descending order. Exact or whole numbers (ie. 26 students in a class) are considered discrete numbers, while those which fall into a given range (ie. 3.6 percent interest rate) are considered continuous numbers. Numerical data is not tied to any specific point in time, they are simply raw numbers.

Arithematic operations can be performed on these dataset. Thus, its meaningful to talk about:

```
2 * height
Cost_price + profit
Total_Cost / total_items
```

Categorical data (or Qualitative data)

Categorical data is sorted by defining characteristics. This can include gender, social class, ethnicity, hometown, the industry you work in, or a variety of other labels. This data type is non-numerical, meaning you are unable to add them together, average them out, or sort them in any chronological order. Categorical data is great for grouping individuals or ideas that share similar attributes, helping your machine learning model streamline its data analysis.
This can be further classified as Nominal and Ordinal data.

Nominal data:

Values have no specific order and can be written in any order.
Examples of such data are:

```
{Male, Female}
{North Zone, South Zone, East Zone, West Zone}
```

Ordinal Data

These are also categorical in nature but the value follows some order. There is a meaning in the order like Very Good would be greater than Good but still these are categorical because we don't know how many times is Very good greater than Good.

Examples:

Course Rating:

Poor	Fair	Good	Very Good	Excellent

Students Grade:

E	D	C	B	A

Preferance:

First Choice	Second Choice	Third Choice

We know the order in the Ordinal data like:

Excellent > Good

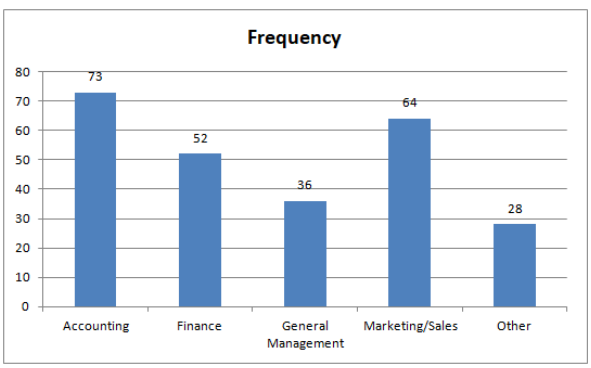
That is order is maintained no matter what value we assign to them.

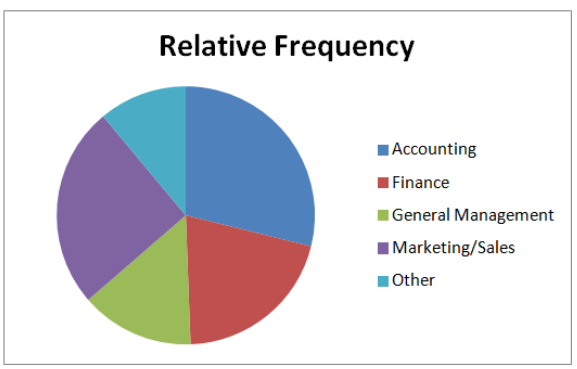
Categorical data can be summarized in a table that lists individual categories and their respective frequency count, e.g. Frequency Distribution.
One can also use a relative frequency distribution which lists the categories and proportion with which each occurs.

Example: Student Placement

Area	Frequency	Relative Frequency
Accounting	73	28.9%
Finance	52	20.6%
General Management	36	14.2%
Marketing/Sales	64	25.3%
Other	28	11.1%
Total	253	100

Frequency distribution and relative frequency distribution can also be summarized as Bar chart and Pie chart respectively, something like below:





Time series data

Time series data consists of data points that are indexed at specific points in time. More often than not, this data is collected at consistent intervals. This makes it easy to compare data from week to week, month to month, year to year, or according to any other time-based metric you desire. The distinct difference between time series data and numerical data is that time series data has established starting and ending points, while numerical data is simply a collection of numbers that aren't rooted in particular time periods.

Examples: *Monthly Sales data for 2019*

January	February	March	April	May	June
45,678	44,987	40,456	51,195	51,561	45,006
July	August	September	October	November	December
46,234	44,839	44,287	42,987	29,839	30,452

Text data

Text data is simply words, sentences, or paragraphs that can provide some level of insight to your machine learning models. Since these words can be difficult for models to interpret on their own, they are most often grouped together or analyzed using various methods such as word frequency, text classification, or sentiment analysis.

How to Choose Statistical Analyses Based on Data Types

So, you understand the different types of data, what you can learn from them, and how to graph them—how else can you use this knowledge? In statistics, the type of variable greatly determines which kinds of analyses you can perform. Discussing these topics is out of scope for this book. Refer Statistics book by us or any book to get more information. We will just get the overview in this section:

- Choosing Hypothesis Tests for Continuous, Binary, and Count Data: Hypothesis tests use sample data to evaluate claims about an entire population. The correct test depends on your variables.
- Chi-squared test of independence when you have two or more categorical variables: This hypothesis test determines whether there is a statistically significant relationship between categorical variables.
- Choosing the Correct Type of Regression Analysis Based on Data Type: Regression analysis describes the relationship between a set of independent variables and a dependent variable. The choice depends on the type of data you have for the dependent variable.

SOURCES FOR EXISTING DATASET

In this section, we will identify good places to find data sets for our work. Whether you want to strengthen your data science portfolio by showing that you can visualize data well, or you have a spare few hours and want to practice your machine learning skills, these dataset will help you.
There is an abundance of places you can find machine learning data, but we have compiled few popular dataset resources to help get you started:

Google's Dataset Search
Google released their Google Dataset Search Engine in September 2018. Use this tool to view datasets across a wide array of topics such as global temperatures, housing market information, or anything else that peaks your interest. Once you enter your search, several applicable datasets will appear on the left side of your screen. Information will be included about each dataset's date of publication, a description of the data, and a link to the data source.

Link: https://datasetsearch.research.google.com/

Microsoft Research Open Data
Microsoft has created a database of free, curated datasets in the form of Microsoft Research Open Data. These are available to the public and are used for "advance state-of-the-art research in areas such as natural language processing, computer vision, and domain specific sciences." Download datasets from published research site or copy them directly to a cloud-based Data Science Virtual Machine.
Link: https://msropendata.com/

Amazon datasets
Amazon Web Services (AWS) has grown to be one of the largest on-demand cloud computing platforms in the world. A plethora of datasets have been made available to the public through AWS resources. These datasets are compiled into Amazon's Registry of Open Data on AWS. Looking up datasets is straightforward, with a search function, dataset descriptions, and usage examples provided.

Link: https://registry.opendata.aws/

UCI Machine Learning Repository
The University of California, School of Information and Computer Science, provides a large amount of information to the public through its UCI Machine Learning Repository database. This database includes nearly 500 datasets, domain theories, and data generators which are used for "the empirical analysis of machine learning algorithms."

Link: https://archive.ics.uci.edu/ml/index.php

Government datasets
The United States Government has released several datasets for public use. These datasets can be used for conducting research, creating data visualizations, developing web/mobile applications, and more. The US Government database can be found at Data.gov and contains information pertaining to industries such as education, ecosystems, agriculture, and public safety, among others. Many countries offer similar databases and most are fairly easy to find.

Apart from these you can also check following sites for dataset:

- SimpleMaps: This has world's cities database including Latitude and Longitude: https://simplemaps.com/data/in-cities
- Kaggle: world's largest data science community with powerful tools and resources to help you achieve your data science goals.

- FiveThirtyEight: one of the best-established data journalism outlets in the world. They also makes the data sets used in its articles available online on Github and on its own data portal.
- Propublica https://www.propublica.org/datastore/datasets/finance
- Quandl is a repository of economic and financial data. Some of this information is free, but many data sets require purchase.
- Data.world is a user-driven data collection site (among other things) where you can search for, copy, analyze, and download data sets. You can also upload your own data to data.world and use it to collaborate with others.
- The World Bank: You can browse world bank data sets directly, without registering. The data sets have many missing values (which is great for cleaning practice), and it sometimes takes several clicks to actually get to data.
- Twitter has a good streaming API, and makes it relatively straightforward to filter and stream tweets.
- GitHub has an API that allows you to access repository activity and code.
- Quantopian is a site where you can develop, test, and optimize stock trading algorithms.
- Wunderground has an API for weather forecasts that is free up to 500 API calls per day. You could use these calls to build up a set of historical weather data, and then use that to make predictions about the weather tomorrow.

WHY WEB SCRAPING USING PYTHON

Python Web scraping is nothing but the process of collecting data from the web. Web scraping in Python involves automating the process of fetching data from the web. In order to fetch the web data, all we need is the URL or the web address that we want to scrape from. The fetched data will be found in an unstructured form. In order to make use of the data or collect useful insights from it, we transform it into a structured form. Once converted into a structured form, we need to store the data for further processing. The whole process is called web scraping.

We all agree to the fact that data has become a commodity in the 21st century, data-driven technologies have experienced a significant rise, and there is an abundance of data generated from different sources on a daily basis. But, how do we collect data in order to make use of it? Let us discuss for what business scenarios web scraping can be used.

- Data Science: For learning Data Science, we need large amounts of data. Web scraping Python can fulfil this requirement.
- Market Research: Before launching a product or service, companies can study the market in advance with the help of web scraping.
- Tracking Competitive Pricing: Web scraping Python can help study the service or product pricing of the competitors to stay ahead in the market.
- Monitoring Brand Value: Web scraping can be used in order to build brand intelligence and monitor how customers feel about a product or a service.

- Lead Generation: With the help of web scraping, businesses can grow their lead generation by gathering contact details of businesses or individuals.

Web Scraping Rules:

Before we start scraping the web, there are some rules that we must follow to avoid legal issues. They are:

- Check the Terms and Conditions of the website before we scrape it. The Legal Use of Data section will have the information about data that we all can use. Usually, the data we scrape should not be used for commercial purposes. Use the text method as shown below. Every website keeps its rules defined in a txt file. We should inspect it to find the things that are allowed and most importantly the things that are not allowed.
- Keep the pace low. If we request for data from the website too aggressively with our bot or our program, it might be considered as spamming. Add wait time in between to make the program behave like a human.
- Use public content only.

HOW TO PERFORM WEB SCRAPING USING PYTHON

Web Scraping Python Workflow:
A Python web scraping project workflow is commonly categorized into three steps:
Step 1: Fetch web pages that we want to retrieve data from
Step 2: Apply web scraping technologies
Step 3: Finally store the data in a structured form.

Setting up Python Web Scraper:

We will be using Python 3 and PyCharm throughout the hands-on. We will be importing two packages as well.

- For performing HTTP requests: **Import Python requests**
- For handling all of the HTML processing: **Import BeautifulSoup from bs4**

DEMO 1: A STEP-BY-STEP GUIDE ON PYTHON WEB SCRAPING A WIKIPEDIA PAGE

We will be scraping the Wikipedia page to fetch the List of Indian Billionaires published by Forbes in the year 2019. Let us start.

STEP 1: FETCH THE WEB PAGE AND CONVERT THE HTML PAGE INTO TEXT WITH THE HELP OF PYTHON REQUEST LIBRARY

```
#import the python request library to query a website
import requests
#specify the url we want to scrape from
Link =
"https://en.wikipedia.org/wiki/List_of_Indian_people_by_net_worth"
#convert the web page to text
Link_text = requests.get(Link).text
print(Link_text)
```

STEP 2: IN ORDER TO FETCH USEFUL INFORMATION, CONVERT LINK_TEXT (WHICH IS OF STRING DATA TYPE) INTO BEAUTIFULSOUP OBJECT. IMPORT BEAUTIFULSOUP LIBRARY FROM BS4

You might have to install bs4 in not done already. Install using:

```
> pip install bs4
```

You will also need to install lxml as we are going to use lxml parser in our example:

```
> pip install lxml
```

You can use inbuilt parser that comes with bs4 like `html.parser`

```
#import BautifulSoup library to pull data out of HTML and XML
files
from bs4 import BeautifulSoup
#to convert Link_text into a BeautifulSoup Object
soup = BeautifulSoup(Link_text, 'lxml')
print(soup)
```

Different parsers will create different parse trees from the same document. The biggest differences are between the HTML parsers and the XML parsers. The differences become clear on non well-formed HTML documents. The moral is just that you should use the parser that works in your particular case.

- html.parser - BeautifulSoup(markup, "html.parser")
    - Advantages: Decent speed, Lenient (as of Python 2.7.3 and 3.2.)
    - Disadvantages: Not very lenient (before Python 2.7.3 or 3.2.2)
- lxml - BeautifulSoup(markup, "lxml")
    - Advantages: Very fast, Lenient

- Disadvantages: External C dependency
- html5lib - BeautifulSoup(markup, "html5lib")
  - Advantages: Extremely lenient, Parses pages the same way a web browser does, Creates valid HTML5
  - Disadvantages: Very slow, External Python dependency

STEP 3: WITH THE HELP OF THE PRETTIFY() FUNCTION, MAKE THE INDENTATION PROPER

```
#make the indentation proper
print(soup.prettify())
```

STEP 4: TO FETCH THE WEB PAGE TITLE, USE SOUP.TITLE

```
#To take a look at the title of the web page
print(soup.title)
#Output: The first title tag will be given out as an output.
#<title>Forbes list of Indian billionaires - Wikipedia</title>

#We want only the string part of the title, not the tags
print(soup.title.string)
```

STEP 5: WE CAN ALSO EXPLORE <A></A> TAGS IN THE SOUP OBJECT

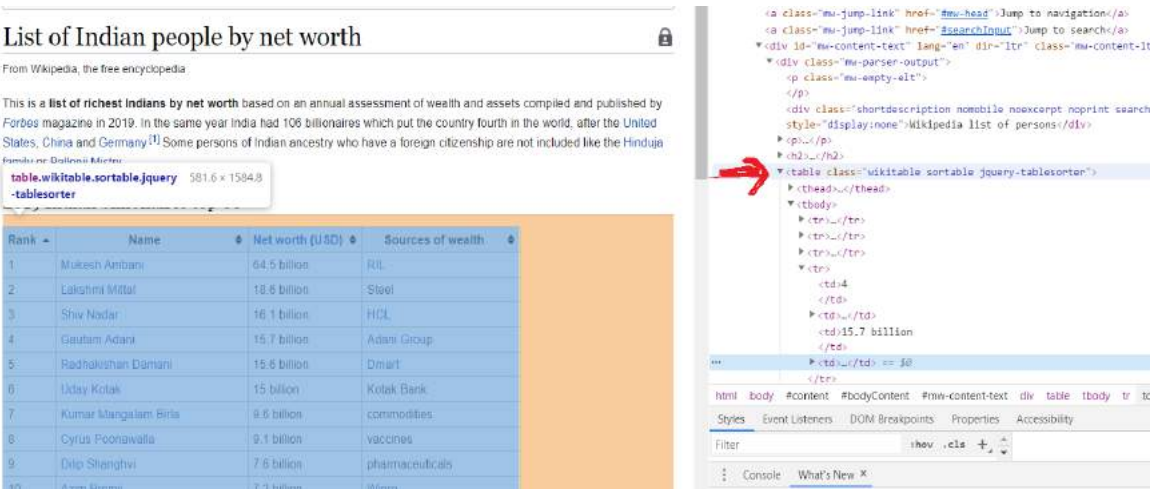
```
#First <a></a> tag
soup.a
#Explore all <a></a> tags
soup.find_all('a')
```

STEP 6: AGAIN, JUST THE WAY WE FETCHED TITLE TAGS, WE WILL FETCH ALL TABLE TAGS

```
#Fetch all the table tags
all_table = soup.find_all('table')
print(all_table)
```

STEP 7: GET THE TABLE CLASS

Since our aim is to get the List of Billionaires from the wiki-page, we need to find out the table class name. Go to the webpage. Inspect the table by placing cursor over the table and inspect the element. Use the keyboard shortcut Ctrl+Shift+I or select Inspect Element from the menu by right-clicking any element of the page. Right-click on the page and select Inspect Element or press F12 on the keyboard.



So, our table class name is 'wikitable sortable'. Let us move ahead and fetch the list.

STEP 8: NOW, FETCH ALL TABLE TAGS WITH THE CLASS NAME 'WIKITABLE SORTABLE'

```
#fetch all the table tags with class name="wikitable sortable"
our_table = soup.find('table', class_= 'wikitable sortable')
print(our_table)
```

STEP 9: FIND ALL <A> TAGS

We can see that the information that we want to retrieve from the table has <a> tags in them. So, find all the <a> tags from table_links.

```
#In the table that we will fetch find the <a> </a>tags
table_links = our_table.find_all('a')
print(table_links)
```

STEP 10: GET THE TITLES

In order to put the title on a list, iterate over table_links and append the title using the get() function

```
#put the title into a list
billionaires = []
for links in table_links:
    billionaires.append(links.get('title'))
print(billionaires)
```

STEP 11: SAVING AS A DATAFRAME

Now that we have our required data in the form of a list, we will be using Python Pandas library to save the data in an Excel file. Before that, we have to convert the list into a DataFrame.
You will have to install pandas if not done before. Install using:

```
> pip install pandas
```

```
#Convert the list into a dataframe
import pandas as pd
df = pd.DataFrame(billionaires)
print(df)
```

STEP 12: USE THE FOLLOWING METHOD TO WRITE DATA INTO AN EXCEL FILE.

You will have to install xlswriter if not done before. Install using:

```
> pip install xlswriter
```

```
#To save the data into an excel file
writer = pd.ExcelWriter('indian_billionaires.xlsx',
engine='xlsxwriter')
df.to_excel(writer, sheet_name='List')
writer.save()
```

Now our data has been saved into an Excel workbook with the name 'indian_billionaires.xlsx' and inside a sheet named 'List'.

STEP 13: JUST TO MAKE SURE IF THE EXCEL WORKBOOK IS SAVED OR NOT, READ THE FILE USING READ_EXCEL

You will have to install xlrd if not done before. Install using:

```
> pip install xlrd
```

```
#check if it's done right or not
df1= pd.read_excel('indian_billionaires.xlsx')
df1
```

We have successfully completed our first web scraping program!

STATISTICS RELEVANT FOR DESCRIPTIVE DATA ANALYSIS

Its not possible to view large dataset with naked eye and make any sense out of it. That's where statistical knowledge comes to our help. Let's look at Statistics concept required to understand the dataset.

Measure of Central Tendency

A measure of central tendency (also referred to as the central location of a distribution) is a summary statistic that represents the center point or typical value of a dataset. These measures indicate where most values in a distribution fall . You can think of it as the tendency of data to cluster around a middle value. In statistics, the three most common measures of central tendency are the mean, median, and mode. Each of these measures calculates the location of the central point using a different method.

Mean: The mean is the arithmetic average. To calculate the mean, just add up all of the values and divide by the number of observations in your dataset.

$$\frac{x_1 + x_2 + \cdots + x_n}{n}$$

Median: The median is the middle value. It is the value that splits the dataset in half. To find the median, order your data from smallest to largest, and then find the data point that has an equal amount of values above it and below it. In the dataset with the odd number of observations, it is simply the number that deivides the dataset into half. Example:

5	7	12	15	21	44	66	76	88

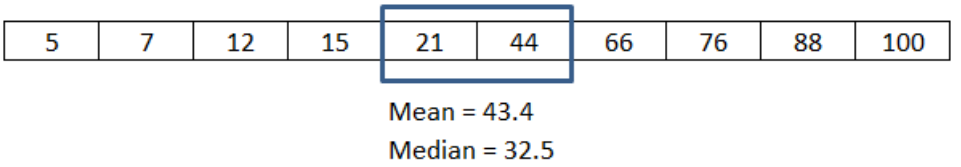
There are 9 numbers in the above dataset and after arranging in the increasing order, 21 divides the data into two equal halves – 4 elements in both the halves.

When there is an even number of values, you count in to the two innermost values and then take the average.

5	7	12	15	21	44	66	76	88	100

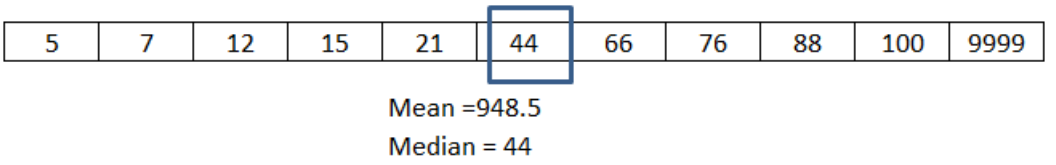
In the above example, the average of 21 and 44 is 32.5. Consequently, 32.5 is the median of this dataset.

Outliers and skewed data have a smaller effect on the median. To understand this, first look at the above data set again:



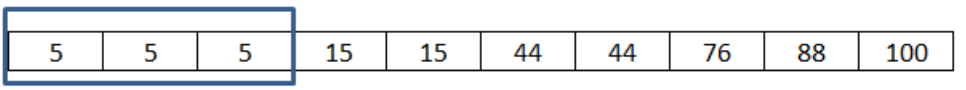
Mean is 43.4 and Median is 32.5

Imagine an outlier value 9999 is added to the dataset. This could most probably be because of error as this number is too way off compared to others. Now let's see the updated Mean and Median. Median shifts to 44 which is not much different compared to orginal value of 32.5 but look at the mean, it changes from 43.4 to 948.5.



Unlike the mean, the median value doesn't depend on all the values in the dataset. Consequently, when some of the values are more extreme, the effect on the median is smaller. Of course, with other types of changes, the median can change. When you have a skewed distribution, the median is a better measure of central tendency than the mean.

Mode: The mode is the value that occurs the most frequently in your data set. On a bar chart, the mode is the highest bar. If the data have multiple values that are tied for occurring the most frequently, you have a multimodal distribution. If no value repeats, the data do not have a mode.



In the dataset above, the value 5 occurs most frequently, which makes it the mode.

Python program to calculate mean, median, mode:

```
#Program 1: Calculate mean, median, mode
#without using any in-built module
# list of elements to calculate mean
num = [5, 5, 5,15, 15, 44, 44, 76, 88, 100]
n = len(num)
#Calculate Mean
get_sum = sum(num)
mean = get_sum / len(num)
print("Mean is: ", mean)

#Calculate Median
num.sort() #Sort the values
if n % 2 == 0:
    median = (num[n//2] + num[n//2 - 1])/2
```

```
else:
    median = num[n//2]
print("Median is: ",median)

#Calculate Mode
Count =[] #empty list to store count
i = 0
while i < n :
    Count.append(num.count(num[i]))
    i += 1
#Save as a dictionary
dict_val = dict(zip(num, Count))
mode_dict = {k for (k, v) in dict_val.items() if v == max(Count)}
print("Mode(s) is/are :", mode_dict)
```

```
#Program 2: Calculate mean, median, mode
#using numpy and scipy module
import numpy
from scipy import stats
num = [5, 5, 5,15, 15, 44, 44, 76, 88, 100]
x = numpy.mean(num)
print("Mean is: ", x)
x = numpy.median(num)
print("Median is: ", x)
x = stats.mode(num)
print(x)
```

Let's apply this knowledge to a problem. Objective of the problem is to find a better school in terms of teaching which helps the students to score higher marks in the exam. In the example below we have taken 2 datasets- list of marks from the two schools and calculate the basic statistics.

```
import numpy
from scipy import stats
school_A = [70,70,71,75, 80, 67,
82,82,82,80,82,70,70,70,70,70,70,70]
school_B = [30,30,41,45,40, 37,
42,32,42,40,100,99,100,100,99,98,99,99]


mean_A = numpy.mean(school_A)
mean_B = numpy.mean(school_B)
med_A = numpy.median(school_A)
med_B = numpy.median(school_B)
mode_A = stats.mode(school_A)
mode_B = stats.mode(school_B)
```

```
print("School A:  Mean is {:0.2f}  Median is {}  Mode is
{}".format(mean_A, med_A, mode_A))
print("School B:  Mean is {:0.2f}  Median is {}  Mode is
{}".format(mean_B, med_B, mode_B))
```

Output:

```
School A:  Mean is 73.94  Median is 70.0  Mode is  ModeResult(mode=array([70]), count=array([9]))
School B:  Mean is 65.17  Median is 43.5  Mode is  ModeResult(mode=array([99]), count=array([4]))
```

In the above example, we see that the difference between the mean values of the 2 schools is not big however the difference between their median is very large. What we learn here is mean or median alone cant define the nature of the data. In order to study the data, we need to put mean and median both into the context before making a decision.

In the continuous data below, no values repeat, which means there is no mode. With continuous data, it is unlikely that two or more values will be exactly equal because there are an infinite number of values between any two values.

Continuous data where no values repeat:

5.1	4.9	5.7	15.2	15.3	44.6	44.9	76	88.1	100.2

This dataset does not have a mode.

When you are working with the raw continuous data, don't be surprised if there is no mode. However, you can find the mode for continuous data by locating the maximum value on a probability distribution plot. If you can identify a probability distribution that fits your data, find the peak value and use it as the mode. Let's see how.

How to graph continuous data

Histograms are a standard way to graph continuous variables because they show the distribution of the values. Let's understand this with an example:

Rajesh is the branch manager at a local bank. Recently, Rajesh has been receiving customer feedback saying that the wait times for a client to be served by a customer service representative are too long. Jeff decides to observe and write down the time spent by each customer on waiting. Here are his findings from observing and writing down the wait times spent by 20 customers:

Customer wait time in second (n=20)	
43.1	42.2
35.6	45.5
37.6	30.3
36.5	31.4
45.3	35.6
43.5	45.2
40.3	54.1
50.2	45.6
47.3	36.5
31.2	43.1

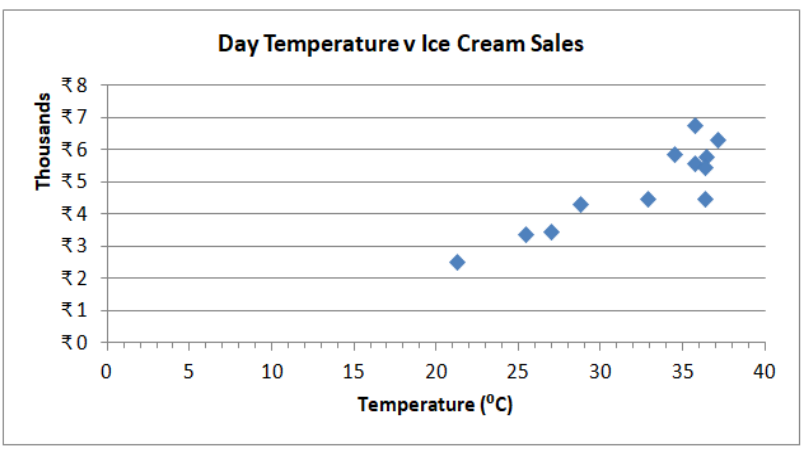
The corresponding histogram with 5-second bins (5-second intervals) would look as above.
We can see that:

- There are 3 customers waiting between 1 and 35 seconds
- There are 5 customers waiting between 1 and 40 seconds
- There are 5 customers waiting between 1 and 45 seconds
- There are 5 customers waiting between 1 and 50 seconds
- There are 2 customers waiting between 1 and 55 seconds

Rajesh can conclude that the majority of customers wait between 35.1 and 50 seconds.

When you have two continuous variables, you can graph them using a scatterplot. A Scatter (XY) Plot has points that show the relationship between two sets of data. Let's take an example to understand this.

Temperature v Ice Cream	
Temperature (°C)	Sales (Rs)
25.4	₹ 3,400
32.8	₹ 4,500
36.4	₹ 5,800
21.2	₹ 2,560
28.7	₹ 4,360
26.9	₹ 3,500
35.7	₹ 5,600
36.3	₹ 5,490
37.1	₹ 6,340
36.3	₹ 4,500
34.4	₹ 5,900
35.7	₹ 6,800

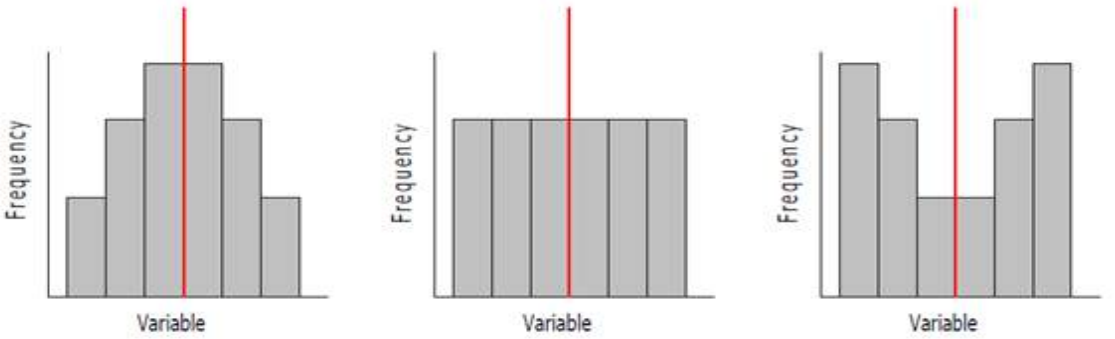


The local ice cream shop keeps track of how much ice cream they sell versus the noon temperature on that day. In the table below are the recorded temperature and the corresponding ice cream sale. It is now easy to see that warmer weather leads to more sales.
Next to that we have plotted the scatter plot as well. In the Unit 8, we will learn to plot these different graphs.

Which is Best—the Mean, Median, or Mode?

When you have a symmetrical distribution for continuous data, the mean, median, and mode are equal. In this case, analysts tend to use the mean because it includes all of the data in the calculations. However, if you have a skewed distribution, the median is often the best measure of central tendency. When you have ordinal data, the median or mode is usually the best choice. For categorical data, you have to use the mode.

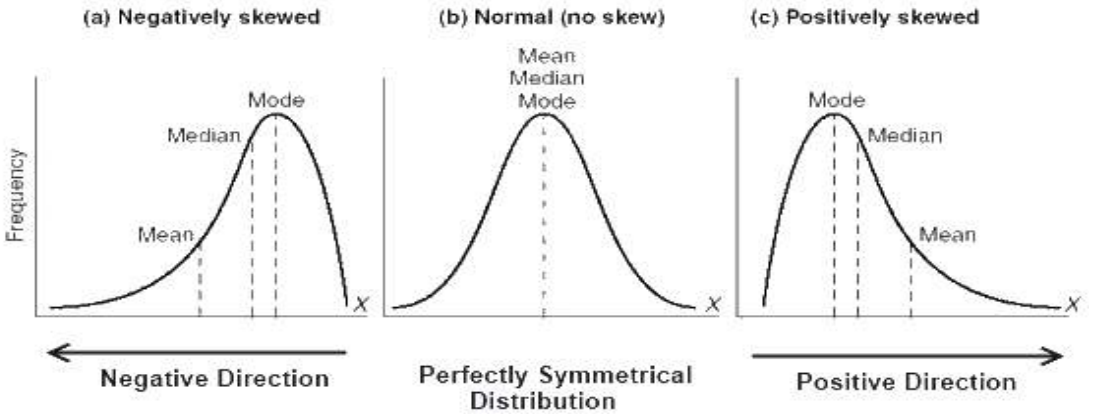
Symmetric Histogram



The histogram displays a symmetrical distribution of data. A distribution is symmetrical if a vertical line can be drawn at some point in the histogram such that the shape to the left and the right of the vertical line are mirror images of each other. In a perfectly symmetrical distribution, the mean and the median are the same. This example has two unimodal (one mode) first two graphs, the mode is the same as the mean and median. The third figure, also a symmetrical distribution, has two modes (bimodal), the two modes are different from the mean and median.

Skewness

A skewness histogram is one with a long tail extending to either the right or the left:



In the above figure, the figure (a) is called negatively skewed as the tail is moving in the –ve direction along the axis. Notice that the mean is less than the median, and they are both less than the mode. The mean and the median both reflect the skewing, but the mean reflects it more so. The figure (b) is a normal distribution with mean, median and mode values being same.

The figure (c) is considered to be positively skewed as the long is moving towards positive direction of the axis. Of the three statistics, the mean is the largest, while the mode is the smallest. Again, the mean reflects the skewing the most. To summarize, generally if the distribution of data is skewed to the left, the mean is less than the median, which is often less than the mode. If the distribution of data is skewed to the right, the mode is often less than the median, which is less than the mean.

Calculating the Skewness:

$a_3 = \sum \frac{(x_i - \overline{x})^3}{ns^3}$, where ***s*** is the sample standard deviation of the data, xi, and $\overline{x}$ is the arithmetic mean and n is the sample size.

The skewness values can be interpreted in the following manner:

- Highly skewed distribution: If the skewness value is less than −1 or greater than +1.
- Moderately skewed distribution: If the skewness value is between −1 and −½ or between +½ and +1 (-1 to -0.5 and +0.5 to +1).
- Approximately symmetric distribution: If the skewness value is between −½ and +½ (-0.5 to +0.5).

Note: In Unit 8, we have explained the applications of these concecpts and also show how to interpret such graphs.

Measures of Variability

A measure of variability is a summary statistic that represents the amount of dispersion in a dataset. How spread out are the values? While a measure of central tendency describes the typical value, measures of variability define how far away the data points tend to fall from the center. We talk about variability in the context of a distribution of values. A low dispersion indicates that the data points tend to be clustered tightly around the center. High dispersion signifies that they tend to fall further away.

Why understanding variability is so essential? Analysts frequently use the mean to summarize the center of a population or a process. Let's take a situation: In a weather report where the

meteorologist shows extreme heat and drought in one area and flooding in another, it would be nice to average those together! But that wouldnt give us right picture, right? While the mean is relevant, people often react to variability even more. When a distribution has lower variability, the values in a dataset are more consistent. However, when the variability is higher, the data points are more dissimilar and extreme values become more likely. Understanding that variability around the mean provides critical information.

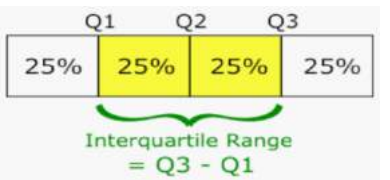
Variability is everywhere. Your commute time to work varies a bit every day. When you order a favorite dish at a restaurant repeatedly, it isn't exactly the same each time. The parts that come off an assembly line might appear to be identical, but they have subtly different lengths and widths. Some variation is inevitable, but problems occur at the extremes. Distributions with greater variability produce observations with unusually large and small values more frequently than distributions with less variability.

Let's understand the concepts of variability before we put them into practice:

Range: The range of a dataset is the difference between the largest and smallest values in that dataset. For example, in the two datasets below, dataset 1 has a range of 100 – 5 = 95 while dataset 2 has a range of 100 – 21 = 79. Dataset 1 has a broader range and, hence, more variability than dataset 2.

Data 1	5	7	12	15	21	44	66	76	88	100
Data 2	50	70	23	35	21	44	66	76	88	100

Interquartile Range: The interquartile range is the middle half of the data. To visualize it, think about the median value that splits the dataset in half. Similarly, you can divide the data into quarters. Statisticians refer to these quarters as quartiles and denote them from low to high as Q1, Q2, and Q3. The lowest quartile (Q1) contains the quarter of the dataset with the smallest values. The upper quartile (Q4) contains the quarter of the dataset with the highest values. The interquartile range is the middle half of the data that is in between the upper and lower quartiles. In other words, the interquartile range includes the 50% of data points that fall between Q1 and Q3. The IQR is the yellow area in the graph below.



Variance: Variance is the average squared difference of the values from the mean. Unlike the previous measures of variability, the variance includes all values in the calculation by comparing each value to the mean. To calculate this statistic, you calculate a set of squared differences between the data points and the mean, sum them, and then divide by the number of observations. Hence, it's the average squared difference.

There are two formulas for the variance depending on whether you are calculating the variance for an entire population or using a sample to estimate the population variance.

Population Variance	Sample Variance
$\sigma^2 = \frac{\sum(X-\mu)^2}{N}$	$s^2 = \frac{\sum(X-M)^2}{N-1}$
σ2 is the population parameter for the variance, μ is the parameter for the population mean, and N is the number of data points, which should include the entire population.	s2 is the sample variance, and M is the sample mean. N-1 in the denominator corrects for the tendency of a sample to underestimate the population variance.

Example of calculating the sample variance

Let's see an example using the formula for a population on a dataset with 15 observations in the table on next page.

Data 1	5	7	12	15	21	44	66	76	88	100
Data 2	50	70	23	35	21	44	66	76	88	100

Data	5	7	12	15	21	44	66	76	88	100	12	15	21	44	66
Mean	39.47	39.47	39.47	39.47	39.47	39.47	39.47	39.47	39.47	39.47	39.47	39.47	39.47	39.47	39.47
Difference	-34.47	-32.47	-27.47	-24.47	-18.47	4.533	26.53	36.53	48.53	60.53	-27.47	-24.47	-18.47	4.533	26.53
Difference Squared	1188	1054	754.4	598.6	341	20.55	704	1335	2355	3664	754.4	598.6	341	20.55	704

Sum of Difference Squared	14434
Population Variance (Sum / # of observations)	962.2

Because the calculations use the squared differences, the variance is in squared units rather the original units of the data. Despite this limitation, various statistical tests use the variance in their calculations. Standard Deviation solves this problem.

Standard Deviation: The standard deviation is the standard or typical difference between each data point and the mean. When the values in a dataset are grouped closer together, you have a smaller standard deviation. On the other hand, when the values are spread out more, the standard deviation is larger because the standard distance is greater.The standard deviation uses the original units of the data, which makes interpretation easier. The standard deviation is the

most widely used measure of variability. For example, in the pizza delivery example, a standard deviation of 5 indicates that the typical delivery time is plus or minus 5 minutes from the mean. It's often reported along with the mean: 20 minutes (s.d. 5).

The standard deviation is just the square root of the variance. Hence, the square root returns the value to the natural units. The symbol for the standard deviation as a population parameter is σ while s represents it as a sample estimate. To calculate the standard deviation, calculate the variance as shown above, and then take the square root of it.
In the above example, discussed under Variance, standard deviation is Square Root of 962.2 = **32**

Python program to calculate these components:
In this program we will use Pandas DataFrame to directly read the data from the github location.
Link of the data: https://github.com/swapnilsaurav/Dataset/blob/master/Mall_Customers.csv
To view the raw data, click on the raw option near top right corner as shown in the figure below:

This will give us raw filelink which we can use to pass to dataframe:
https://raw.githubusercontent.com/swapnilsaurav/Dataset/master/Mall_Customers.csv

```
import pandas as pd
url =
'https://raw.githubusercontent.com/swapnilsaurav/Dataset/master/Ma
ll_Customers.csv'
df = pd.read_csv(url, error_bad_lines=False)
print(df)
#Calculating Standard Deviation for all the numeric columns
print("Standard Deviation: \n",df.std())
#Calculation SD for Specific Columns
print("Standard Deviation in Age Data:
{:.2f}".format(df.loc[:,"Age"].std()))

#Calculating variance
print("Variance in Age Data:
{:.2f}".format(df.loc[:,"Age"].var()))
```

```
#IQR can be calculated using the iqr() function
from scipy.stats import iqr
print("IQR of Age Data: ",iqr(df["Age"]))

#Calculating Skewness
print("Skewness of Age Data: {:.2f}".format(df["Age"].skew()))
```

Sample Output

```
Standard Deviation:
 CustomerID                57.879185
Age                        13.969007
Annual Income (k$)         26.264721
Spending Score (1-100)     25.823522
dtype: float64
Standard Deviation in Age Data: 13.97
Variance in Age Data:  195.13
IQR of Age Data:  20.25
Skewness of Age Data: 0.49
```

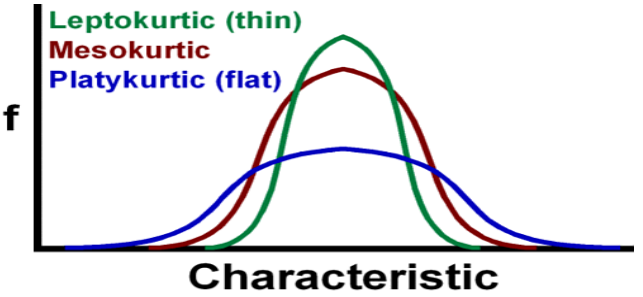
Which is Best - the Range, Interquartile Range, Variance or Standard Deviation?
Variance is in squared units and hence doesn't provide an intuitive interpretation. Now looking at the other three options

When you are comparing samples that are the same size, consider using the range as the measure of variability. It's a reasonably intuitive statistic. Just be aware that a single outlier can throw the range off. The range is particularly suitable for small samples when you don't have enough data to calculate the other measures reliably, and the likelihood of obtaining an outlier is also lower.

When you have a skewed distribution, the median is a better measure of central tendency, and it makes sense to pair it with either the interquartile range or other percentile-based ranges because all of these statistics divide the dataset into groups with specific proportions.

For normally distributed data, or even data that aren't terribly skewed, using the tried and true combination reporting the mean and the standard deviation is the way to go. This combination is by far the most common. You can still supplement this approach with percentile-base ranges as you need.

Kurtosis



Kurtosis is a statistical measure that defines how heavily the tails of a distribution differ from the tails of a normal distribution. In other words, kurtosis identifies whether the tails of a given distribution contain extreme values. Along with skewness, kurtosis is an important descriptive statistic of data distribution. However, the two concepts must not be confused with each other. Skewness essentially measures the symmetry of the distribution, while kurtosis determines the heaviness of the distribution tails.

An excess kurtosis is a metric that compares the kurtosis of a distribution against the kurtosis of a normal distribution. The kurtosis of a normal distribution equals 3. Therefore, the excess kurtosis is found using the formula below:

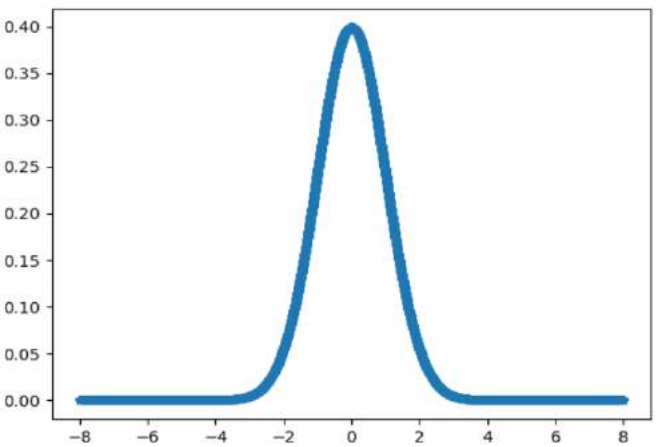
Excess Kurtosis = Kurtosis – 3

The types of kurtosis are determined by the excess kurtosis of a particular distribution. The excess kurtosis can take positive or negative values, as well as values close to zero. Distributions with kurtosis less than 3 are said to be platykurtic and Distributions with kurtosis greater than 3 are said to be leptokurtic.

Mesokurtic	Leptokurtic	Platykurtic
Data that follows a mesokurtic distribution shows an excess kurtosis of zero or close to zero. This means that if the data follows a normal distribution, it follows a mesokurtic distribution	Leptokurtic indicates a positive excess kurtosis. The leptokurtic distribution shows heavy tails on either side, indicating large outliers. In finance, a leptokurtic distribution shows that the investment returns may be prone to extreme values on either side. Therefore, an investment whose returns follow a leptokurtic distribution is considered to be risky.	A platykurtic distribution shows a negative excess kurtosis. The kurtosis reveals a distribution with flat tails. The flat tails indicate the small outliers in a distribution. In the finance context, the platykurtic distribution of the investment returns is desirable for investors because there is a small probability that the investment would experience extreme returns.

Python program to calculate Kurtosis

```
# Graph using numpy.linspace()
# finding kurtosis

from scipy.stats import kurtosis
import numpy as np
import pylab as p

x1 = np.linspace( -8, 8, 5000 )
y1 = 1./(np.sqrt(2.*np.pi)) * np.exp( -.5*(x1)**2 )

p.plot(x1, y1, '*')
p.show()
#; Fisher's definition is used (normal 0.0)
# if True; else Pearson's definition is used - Default
# (normal 3.0) if set to False.

print('Kurtosis for normal distribution :', kurtosis(y1))
print('Kurtosis for normal distribution :',    kurtosis(y1, fisher
= False))
```



Frequency Distribution

Univariate frequency tables

Rank	Degree of agreement	Number
1	Strongly agree	20
2	Agree somewhat	30
3	Not sure	20
4	Disagree somewhat	15
5	Strongly disagree	15

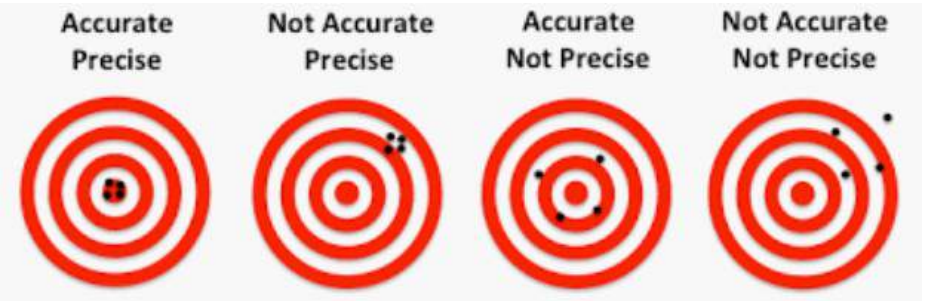
Joint frequency distributions

Bivariate joint frequency distributions

	Dance	Sports	TV	Total
Men	2	10	8	20
Women	16	6	8	30
Total	18	16	16	50

- Managing and operating on frequency tabulated data is much simpler than operation on raw data. This gives you power to analyze large data.
- Statistical hypothesis testing is founded on the assessment of differences and similarities between frequency distributions. This assessment involves measures of central tendency or averages, such as the mean and median, and measures of variability or statistical dispersion, such as the standard deviation or variance.
- A frequency distribution is said to be skewed when its mean and median are different.
- The kurtosis of a frequency distribution is the concentration of scores at the mean, or how peaked the distribution appears if depicted graphically in a histogram.
- If the distribution is more peaked than the normal distribution it is said to be leptokurtic; if less peaked it is said to be platykurtic.

Accuracy versus Precision



Accuracy refers to the closeness of a measured value to a standard or known value. For example, if in lab you obtain a weight measurement of 3.2 kg for a given substance, but the actual or known weight is 10 kg, then your measurement is not accurate. In this case, your measurement is not close to the known value. Precision refers to the closeness of two or more measurements to each other. Using the example above, if you weigh a given substance five times, and get 3.2 kg each time, then your measurement is very precise. Precision is independent of accuracy. You can be very precise but inaccurate, as described above. You can also be accurate but imprecise.

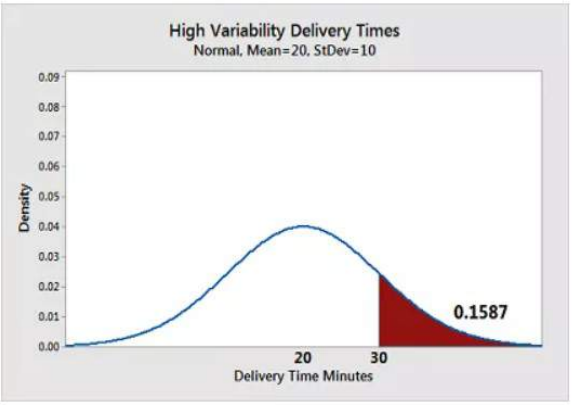
Let's say you know your true height is exactly 5'6". You measure yourself and get 5'0". Again you measure yourself, and this time you get 6'0". You take average, and you get a accurate value of 5'6" but your values are not precise as their readings don't match.

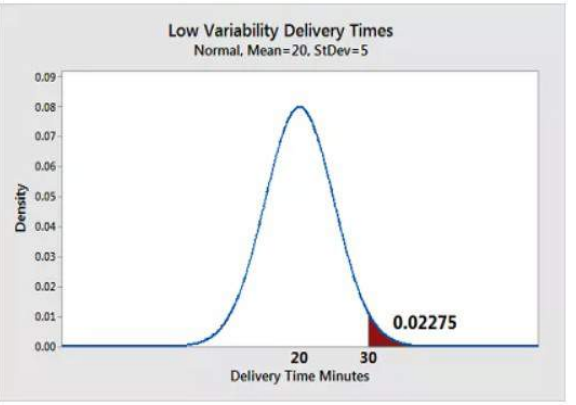
In practical purpose, you have to be accurate and precise then your process is best.

Application

Now that we understand different components, let's take an example to analyze:
We take an example of delivery times as advertised by two hypothetical quick service restaurants (QSRs):

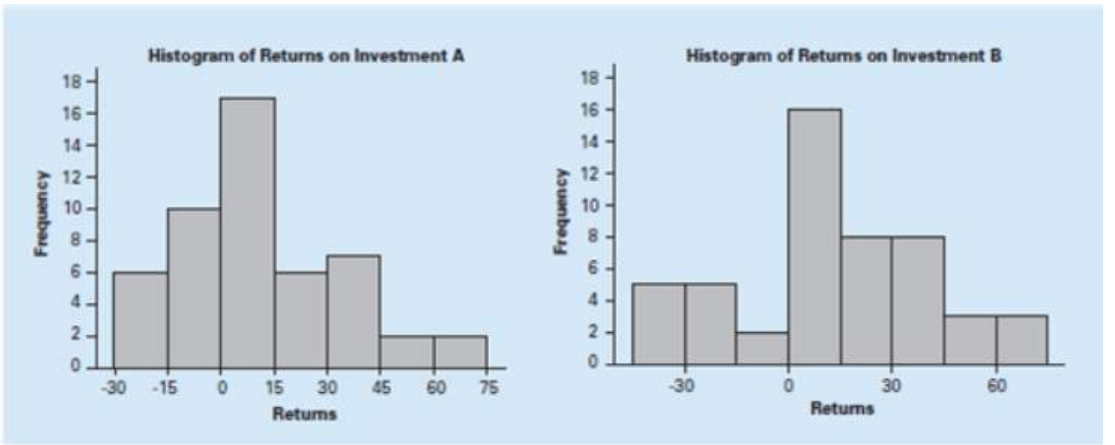
Both restaurants look good by their mean value. However, this equivalence can be deceptive! To determine the restaurant that you should order from when you're hungry, we need to analyze their variability.





The graphs above display the distribution of delivery times and provide the answer. The restaurant with more variable delivery times has the broader distribution curve. The time to serve by first restaurant varies by a large range (higher standard deviation) compared to the second restaurant. Hence, we can trust second retaurant more with their mean time than the first one.

Let's take a look at another example:



Suppose you are facing a decisiom about where to invest that small fortune that remains after deducting your anticipated expenses. A friend has suggested two investment options where the rate of return is depicted in the above histogram image. Which one would you prefer to invest?

Interpretation of the graph:

1. The center of the histogram of returns of investment A is slightly lower than that for investment B.
2. The spread of the returns for investment A is considerably less than that of B
3. Both histograms are slightly positive skewed.

These findings suggest that investment A is superior. Although the returns of A are slightly less than those from B, the wider spread for B makes it unappealing to most investors. Since both the histograms are positively skewed, both investments allow for the possibility of a relatively large return.

Now that we know how and from where to get the data and also understand the nature and type of data, in the next chapter we will focus on cleaning of the data before interpreting the result.

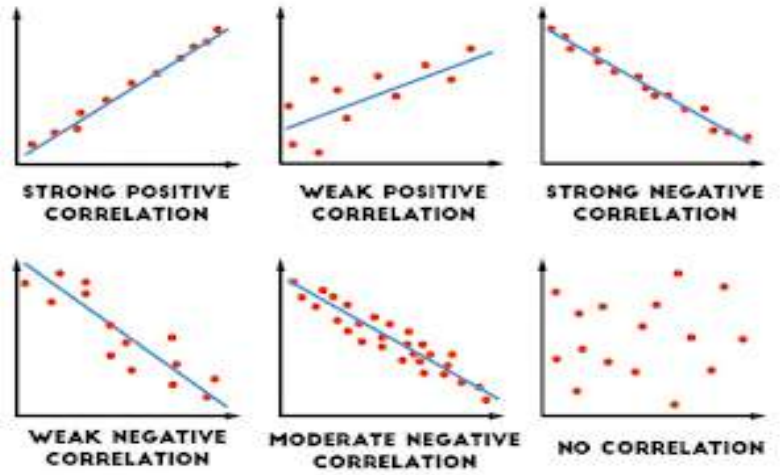
Correlation

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related. For example, height and weight are related; taller people tend to be heavier than shorter people. The relationship isn't perfect. People of the same height vary in weight, and you can easily think of two people you know where the shorter one is heavier than the taller one. Nonetheless, the average weight of people 5'5'' is less than the average weight of people 5'6'', and their average weight is less than that of people 5'7'', etc. Correlation can tell you just how much of the variation in peoples' weights is related to their heights. Although this correlation is fairly obvious your data may contain unsuspected correlations. You may also suspect there are correlations, but don't know which are the strongest. An intelligent correlation analysis can lead to a greater understanding of your data.

Like all statistical techniques, correlation is only appropriate for certain kinds of data. Correlation works for quantifiable data in which numbers are meaningful, usually quantities of some sort. It cannot be used for purely categorical data (nominal data), such as gender, brands purchased, or favorite color.

Correlation Coefficient

The main result of a correlation is called the correlation coefficient (or "r"). It ranges from -1.0 to +1.0. The closer r is to +1 or -1, the more closely the two variables are related.



If r is close to 0, it means there is no relationship between the variables. If r is positive, it means that as one variable gets larger the other gets larger. If r is negative it means that as one gets larger, the other gets smaller (often called an "inverse" correlation).

While correlation coefficients are normally reported as r = (a value between -1 and +1), squaring them makes then easier to understand. The square of the coefficient (or r square) is equal to the percent of the variation in one variable that is related to the variation in the other. After squaring r, ignore the decimal point. An r of .5 means 25% of the variation is related (.5 squared =.25). An r value of .7 means 49% of the variance is related (.7 squared = .49).

Let's now look at the below set of formulae:

$$\mathbf{VAR(X)} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - E(X))^2 \qquad (1)$$

$$\mathbf{COV(X,\ Y)} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - E(X))(y_i - E(Y)) \qquad (2)$$

$$\mathbf{COR(X,\ Y)} = \frac{\mathbf{COV(X,\ Y)}}{\sqrt{\mathbf{VAR(X)VAR(Y)}}} \qquad (3)$$

$$\mathbf{R}^2 = 1 - \frac{\mathbf{VAR(X,\ Y)}_{FittedLine}}{\mathbf{VAR(X,\ Y)}_{Mean}} \qquad (4)$$

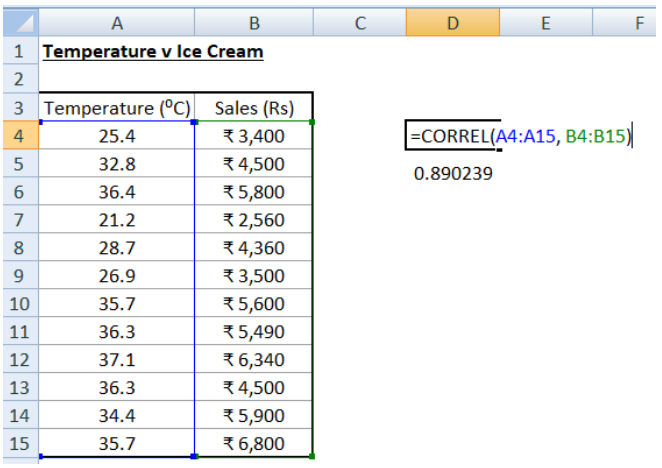
Formula (1) calculates variance, which we have already seen before in this chapter.

Formula (2) talks about Covariance. Covariance is a measure used to determine how much two variables change in tandem. The unit of covariance is a product of the units of the two variables. Covariance is affected by a change in scale. The value of covariance lies between -∞ and +∞.

Formula (3) shows how to calculate correlation. It shows it dependent on covariance. In simple words, both the terms- Covariance and Correlation, measure the relationship and the dependency between two variables. “Covariance” indicates the direction of the linear relationship between variables. “Correlation” on the other hand measures both the strength and direction of the linear relationship between two variables. Correlation is a function of the covariance. What sets them apart is the fact that correlation values are standardized whereas, covariance values are not. You can obtain the correlation coefficient of two variables by dividing the covariance of these variables by the product of the standard deviations of the same values. When you divide the covariance values by the standard deviation, it essentially scales the value down to a limited range of -1 to +1. This is precisely the range of the correlation values.

R-squared (shown in Formula 4) is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. 0% indicates that the model explains none of the variability of the response data around its mean. Here we only introduce the concept; discussion on R-squared is out of scope for this book.

Lets take the previous example of Ice cream sale versus the temperature and calculate the correlation factor:



	A	B	C	D	E	F
1	**Temperature v Ice Cream**					
2						
3	Temperature (°C)	Sales (Rs)				
4	25.4	₹ 3,400		=CORREL(A4:A15, B4:B15)		
5	32.8	₹ 4,500		0.890239		
6	36.4	₹ 5,800				
7	21.2	₹ 2,560				
8	28.7	₹ 4,360				
9	26.9	₹ 3,500				
10	35.7	₹ 5,600				
11	36.3	₹ 5,490				
12	37.1	₹ 6,340				
13	36.3	₹ 4,500				
14	34.4	₹ 5,900				
15	35.7	₹ 6,800				

Calculate Correlation factor using Excel:

Select a blank cell that you will put the calculation result, enter this formula =CORREL(A4:A15,B4:B15), where the cell numbers are where you data is present, and press Enter key to get the correlation coefficient. See screenshot as shown above. The result is 0.89 which means there is a strong positive correlation between the two dataset.

A key thing to remember when working with correlations is never to assume a correlation means that a change in one variable causes a change in another. Sales of personal computers and athletic shoes have both risen strongly over the years and there is a high correlation between them, but you cannot assume that buying computers causes people to buy athletic shoes (or vice versa).

The second caveat is that the Pearson correlation technique works best with linear relationships: as one variable gets larger, the other gets larger (or smaller) in direct proportion. It does not work well with curvilinear relationships (in which the relationship does not follow a straight line). An example of a curvilinear relationship is age and health care. They are related, but the relationship doesn't follow a straight line. Young children and older people both tend to use much more health care than teenagers or young adults. Multiple regression can be used to examine curvilinear relationships, but that is outside the scope of this book.

Python Program to find Correlation coefficient

```
# calculate the covariance between two variables
from numpy import cov
temp_data = [25.4, 32.8, 36.4, 21.2, 28.7, 26.9, 35.7, 36.3, 37.1,
36.3, 34.4, 35.7]
sales_data = [3400, 4500, 5800, 2560, 4360, 3500, 5600, 5490,
6340, 4500, 5900, 6800]
# calculate covariance matrix
covariance = cov(temp_data, sales_data)
print(covariance)
#print('Covariance: %.2f' % covariance)

from scipy.stats import spearmanr, pearsonr
# calculate spearman's correlation
corr, pvalue = spearmanr(sales_data, temp_data)
print('Spearmans correlation: %.2f' % corr)
# calculate pearson's correlation
corr, pvalue = pearsonr(sales_data, temp_data)
print('Pearsons correlation: %.2f' % corr)

import matplotlib.pyplot as plt
plt.scatter(temp_data, sales_data)
plt.show()
```

Sample Output

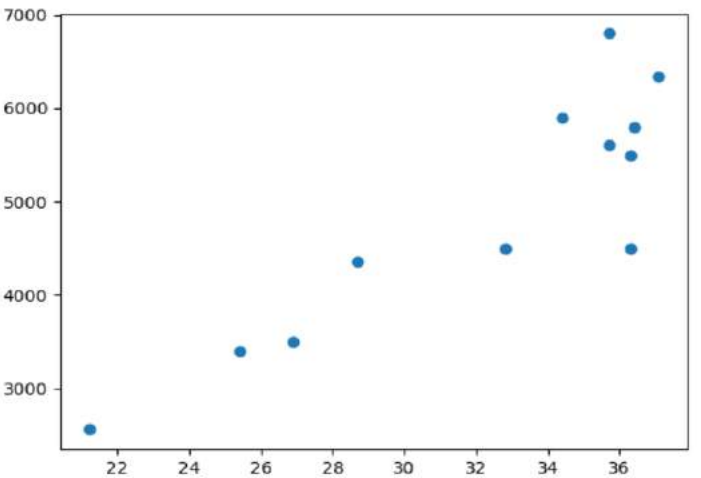
```
[[2.84117424e+01 6.17518939e+03]
 [6.17518939e+03 1.69351742e+06]]
Spearmans correlation: 0.78
Pearsons correlation: 0.89
```

The 2x2 array returned by np.cov(a,b) has elements equal to:

```
cov(a,a)  cov(a,b)

cov(a,b)  cov(b,b)
```

- The Pearson Product Moment Correlation tests the linear relationship between two continuous variables. Linear means a relationship when two variables change in the same direction at a constant rate. Excel performs Pearson Correlation.
- Spearman Rank Correlation evaluates the monotonic relationship between the ranked values. In a monotonic relationship, the variables also tend to change together, but not necessarily at a constant rate.

Plot:



Note: In this chapter (and the book), author has covered the Basic Descriptive Statistics. To know more about the statistics for Machine Learning, please refer to our book:

Essential Probability & Statistics for Machine Learning (Implementation using Python)

By Swapnil Saurav

Now available world-wide.

UNIT 4: CLEAN YOUR DATA

There's a good chance you've ran experiments or you will be working on the Iris Dataset for demonstrating basic machine learning models. It's a good dataset for educational purposes, but all too often I've seen aspiring Data Scientists grow frustrated when they attempt to apply those skills to real-world problems. Why are projects from academia so different from those faced in the real-world?

There is a disconnect between applying your skills in a competition vs. corporate projects. Before focusing on result, considerable amount of time has to be given to cleaning the date (covered in this chapter) and Preparing the data (next chapter).

Data cleaning or cleansing is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

You can understand the importance of data cleaning as some companies have entire teams devoted to cleaning code. Data scientists spend a large amount of their time cleaning datasets and getting them down to a form with which they can work. According to IBM Data Analytics you can expect to spend up to 80% of your time cleaning data.

In this chapter we will cover following topics:

- Missing Data
- Irregular Data (Outliers)
- Unnecessary Data — Repetitive Data, Duplicates and more
- Inconsistent Data — Capitalization, Addresses and more
- Wrong format Data – e.g. Date as String

We have divided this chapter into 2 parts: Data Wrangling where we will learn some common libraries in Python used to handle the data and in the last section, we will see applications.

DATA WRANGLING USING NUMPY

Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics. A data wrangler is a person who performs these transformation operations.

This may include further munging, data visualization, data aggregation, training a statistical model, as well as many other potential uses. Data munging as a process typically follows a set of general steps which begin with extracting the data in a raw form from the data source,

"munging" the raw data using algorithms (e.g. sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and future use.

Some of the steps involved are:

- Data cleaning: Data cleansing or data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Data cleansing may be performed interactively with data wrangling tools, or as batch processing through scripting.
- Data editing: Correcting errors in a corpus of data. Data editing is defined as the process involving the review and adjustment of collected survey data. The purpose is to control the quality of the collected data. Data editing can be performed manually, with the assistance of a computer or a combination of both.
- Data scraping, extracting parts of a corpus of data with automated tools. Data scraping is a technique in which a computer program extracts data from human-readable output coming from another program.
- Data curation, a more general and abstract activity. Data curation is the organization and integration of data collected from various sources. It involves annotation, publication and presentation of the data such that the value of the data is maintained over time, and the data remains available for reuse and preservation. Data curation includes "all the processes needed for principled and controlled data creation, maintenance, and management, together with the capacity to add value to data". In science, data curation may indicate the process of extraction of important information from scientific texts, such as research articles by experts, to be converted into an electronic format, such as an entry of a biological database.
- Data pre-processing, a step of cleaning data in data mining for analysis purposes. Data preprocessing is an important step in the data mining/analytics process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Data-gathering methods are often loosely controlled, resulting in out-of-range values (e.g., Income: −100), impossible data combinations (e.g., Sex: Male, Pregnant: Yes), missing values, etc. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis. Often, data preprocessing is the most important phase of a machine learning projects. If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time. Data preprocessing includes cleaning, Instance selection, normalization, transformation, feature extraction and selection, etc. The product of data preprocessing is the final training set. Data pre-processing may affect the way in which outcomes of the final data processing can be interpreted. *Note: We have dealt this topic in detail under MachineLearning (Section E)->SciKit Learn package.*
- Data fusion and data integration: Data fusion is the process of integrating multiple data sources to produce more consistent, accurate, and useful information than that provided

by any individual data source. Humans are a prime example of Data Fusion. As humans, we rely heavily on our senses such as our Vision, Smell, Taste, Voice and Physical Movement. A combination of all these senses combine on a daily basis to help us in performing most if not all tasks in our day to day lives. Data integration involves combining data residing in different sources and providing users with a unified view of them.

- Data preparation: Data preparation is the act of manipulating (or pre-processing) raw data (which may come from disparate data sources) into a form that can readily and accurately analysed, e.g. for business purposes. Data preparation is the first step in data analytics projects and can include many discrete tasks such as loading data or data ingestion, data fusion, data cleansing, data augmentation, and data delivery.
- OpenRefine: OpenRefine, formerly called Google Refine and before that Freebase Gridworks, is a standalone open source desktop application for data cleanup and transformation to other formats, the activity known as data wrangling. It is similar to spreadsheet applications (and can work with spreadsheet file formats); however, it behaves more like a database. Unlike spreadsheets, no formulas are stored in the cells, but formulas are used to transform the data, and transformation is done only once. Transformation expressions can be written in General Refine Expression Language (GREL), Jython (i.e. Python) and Clojure. It is available for download and use on the local machine as a web interface. When starting OpenRefine, it starts a web server and starts a browser to open the web UI powered by this web server.
- Semantic mapping (data integration)
- Simultaneous editing, efficient repeated edition of text in a multiple selection through direct manipulation.
- Extract, transform, load
- Big Data

In this chapter, we will learn how to handle data using 3 modules: Numpy, Scipy and Pandas. Numpy is required by pandas (and by virtually all numerical tools for Python). Scipy is not strictly required for pandas but is listed as an "optional dependency". Pandas is not an alternative to Numpy and/or Scipy. Rather, it's an extra tool that provides a more streamlined way of working with numerical and tabular data in Python. You can use pandas data structures but freely draw on Numpy and Scipy functions to manipulate them.

- Numpy is written in C and use for mathematical or numeric calculation.
- It is faster than other Python Libraries
- Numpy is the most useful library for Data Science to perform basic calculations.
- Numpy contains nothing but array data type which performs the most basic operation like sorting, shaping, indexing, etc.

Let’s practice Numpy

NUMPY

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with the arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of non-negative integers. In NumPy dimensions are called axes. For example, the coordinates of a point in 3D space [1, 2, 1] has one axis. That axis has 3 elements in it, so we say it has a length of 3. In the example pictured below, the array has 2 axes. The first axis has a length of 2, the second axis has a length of 3.

```
[[1 2 3]
 [4 5 6]]
```

Arrays

NumPy's array class is called ndarray. It is also known by the alias array. Note that numpy.array is not the same as the Standard Python Library class array.array, which only handles one-dimensional arrays and offers less functionality. A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension. The more important attributes of an ndarray object are:

ndarray.ndim	the number of axes (dimensions) of the array.
ndarray.shape	the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.
ndarray.size	the total number of elements of the array. This is equal to the product of the elements of shape.
ndarray.dtype	an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.
ndarray.itemsize	the size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize.
ndarray.data	the buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

We can initialize numpy arrays from nested Python lists, and access elements using square brackets:

```
import numpy as np

x = range(16)
x = np.reshape(x,(4,4))
print(x)

a = np.array([1, 2, 3])   # Create a rank 1 array (Single
row)
print(a)                  # Prints "[1, 2, 3]"
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b)
print(b.shape)                     # Prints "(2, 3)" 2 rows 3
columns
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Numpy also provides many functions to create arrays:

```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)
b = np.ones((1,2))

c = np.full((2,2), 7)  # Create a constant array
print(c)
d = np.eye(2)         # Create a 2x2 identity matrix
print(d)

e = np.random.random((2,2))
print(e)
```

Output

```
[[0. 0.]
 [0. 0.]]
[[7 7]
 [7 7]]
[[1. 0.]
 [0. 1.]]
[[0.89403954 0.24898726]
 [0.9919712  0.52524809]]
```

Array indexing
Numpy offers several ways to index into arrays.

Slicing: Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:

```
import numpy as np

# Create a rank 2 array with shape (3, 4)
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a)
# Use slicing to pull out the subarray consisting of the
first 2 rows
# and columns 1 and 2; b is the following array of shape (2,
2):
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so
modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0,
1]
print(a[0, 1])   # Prints "77"

#You can also mix integer indexing with slice indexing.
# Two ways of accessing the data in the middle row of the
array.
# Mixing integer indexing with slices yields an array of
lower rank,
# while using only slices yields an array of the same rank as
the
# original array:
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)

# We can make the same distinction when accessing columns of
an array:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)
print(col_r2, col_r2.shape)
```

Datatypes

Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that you can use to construct arrays. Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype. Here is an example:

```
import numpy as np

# Let numpy choose the datatype
x = np.array([1, 2])
print(x.dtype)  # Prints "int32"
# Force a particular datatype
x = np.array([1, 2], dtype=np.int64)
print(x.dtype)   # Prints "int64"

# Let numpy choose the datatype
x = np.array([1.0, 2.0])
print(x.dtype) # Prints "float64"
```

Array math

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:

```
import numpy as np
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array
print(x * y)
print(np.multiply(x, y))

# Matrix Multiplication; both produce the array
print(x @ y)
print(np.matmul(x, y))

#Dot Multiplication
print(x.dot(y))

# Elementwise division; both produce the array
```

```
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
print(np.sqrt(x))
```

Dot Product:

For 2-D vectors, it is the equivalent to matrix multiplication. For 1-D arrays, it is the inner product of the vectors. For N-dimensional arrays, it is a sum product over the last axis of a and the second-last axis of b.

Examples to Practice

```
import numpy as np

A = np.arange(3)
print("A: \n",A)
A = np.arange(10)**3
print(A)
print("Slicing: ",A[2:5])
#Similar to String or List slicing
#Iterating through the Array
print("Calculating cube root of the values:")
for i in A:
    print(i**(1/3))

c = np.array(
    [[[0, 1, 2],  # a 3D array (two stacked 2D arrays)
    [10, 12, 13]],
    [[100, 101, 102],
    [110, 112, 113]]])
print("Shape of the matrix is: ", np.shape(c))
print(c[1,...])
print("Matrix C: \n",c.reshape(6,2))

#Iterating in Flat format
print("Flat Format:")
for element in c.flat:
    print(element)
```

More Operations using Numpy

```
import numpy as np

#Splitting one array into several smaller ones
A = np.arange(9)**3
A.resize(3,3)
```

```
print("Original Matrix: \n",A)
# Split a into 3
print("Horizontal Split into 3: ",np.hsplit(A,3))
print("Vertical Split into 3: ",np.vsplit(A,3))
```

Output:

```
Original Matrix:
 [[  0   1   8]
 [ 27  64 125]
 [216 343 512]]
Horizontal Split into 3:  [array([[  0],
       [ 27],
       [216]], dtype=int32), array([[  1],
       [ 64],
       [343]], dtype=int32), array([[  8],
       [125],
       [512]], dtype=int32)]
Vertical Split into 3:  [array([[0, 1, 8]], dtype=int32), array([[ 27,  64, 125]], dtype=int32), array([[216, 343, 512]], dtype=int32)]
```

Copy and View in Numpy

```
import numpy as np

a = np.array([[ 0,  1,  2,  3],
              [ 4,  5,  6,  7],
              [ 8,  9, 10, 11]])
b = a    # no new object is created
print(b is a)  # a and b are two names for the same ndarray object
print("ID of a: ",id(a))
print("ID of b: ",id(b))
#Value you will get may vary but you will see both has same value
c = a.view()
print(c is a)
# output: False as c is a view of the data owned by a
print("Checking base: ",c.base is a)
# Output: True as c is a view of the data owned by a
#Deep Copy using copy
d = a.copy()
print(c is a)
# output: False as d is a new array object
print("Checking base: ",c.base is a)
# Output: False as d doesn't share anything with a
```

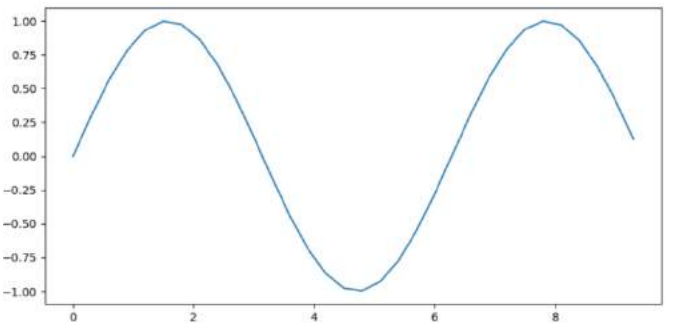
Output:

```
True
ID of a:  50935512
ID of b:  50935512
False
Checking base:  True
False
Checking base:  True
```

Special Function

Plot a Sin graph using Numpy:

```
import numpy as np
import matplotlib.pyplot as plt
x= np.arange(0, 3*np.pi,0.3)
y = np.sin(x)
plt.plot(x,y)
plt.show()
```



LINEAR ALGEBRA: MATRIX APPLICATIONS USING NUMPY

```
#Program to print Identity Matrix of order 'n', where n is an integer
import numpy as np
# 2x2 matrix with 1's on main diagnol
b = np.identity(2, dtype = float)
print("Matrix b : \n", b)
a = np.identity(4)
print("\n Matrix a : \n", a)
```

- An Identity Matrix is a **square matrix** whose main diagonal elements are **1** and all the other elements are **0**.
- Identity Matrix is also called as **Unit Matrix.**

Example: $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Python Program to Find the Transpose of a Matrix

Transpose of a matrix is the interchanging of rows and columns. It is denoted as X'. The element at ith row and jth column in X will be placed at jth row and ith column in X'. So if X is a 3x2 matrix, X' will be a 2x3 matrix.

```
import numpy
matrix=[[1,2,3],[4,5,6]]
print(matrix)
print("\n")
print(numpy.transpose(matrix))
```

Matrix Multiplication

- Two matrices A and B may be multiplied only if matrix A and matrix B obey following rule:
  - The number of columns of A **equals** the number of rows of B.
- In general, if A is an m × n matrix and B is an n × p matrix, then A × B is possible and the order of the resulting product matrix AB will be **m × p**.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} = \begin{bmatrix} (aj+bm+cp) & (ak+bm+cq & (al+bo+cr) \\ (dj+em+fp) & (dk+en+fq) & (dl+eo+fr) \\ (gj+hm+ip) & (gk+hn+iq) & (gl+ho+ir) \end{bmatrix}$$

```
# Python code to demonstrate matrix operations
# mul()
# importing numpy for matrix operations
import numpy
# initializing matrices
x = numpy.array([[10, 20, 50], [5, 8, 8]])
y = numpy.array([[9, 8, 9], [4, 3, 2]])
# using mul()  multiply matrices
print ("The element wise multiplication of matrix is : ")
print (numpy.multiply(x,y))
```

Inverse of a Matrix

```
# Python Program to Compute Determinant of a Matrix
import numpy as np
b = np.array([[6,1,1], [4, -2, 5], [2,8,7]])
print(b)
print(np.linalg.det(b))
#alternative: manually calculating
print(6*(-2*7 - 5*8) - 1*(4*7 - 5*2) + 1*(4*8 - -2*2))
```

- Determinant of a Matrix is a special number that is defined only for **Square Matrix**.
- The **symbol** for determinant is two vertical lines either side.

  Example: **|A|** means the determinant of the matrix **A**

Determinant of 2 X 2 Matrix

$$M = \begin{bmatrix} p & q \\ r & s \end{bmatrix}$$

$$\det(M) = (p*s)-(q*r)$$

Inverse of a Matrix

- For a square matrix **A**, the inverse is written $\mathbf{A^{-1}}$. When **A** is multiplied by $\mathbf{A^{-1}}$ the result is the **Identity matrix** 'I'.
- Non-square matrices do not have inverses.

$$\mathbf{A\,A^{-1} = A^{-1}\,A = I}$$

For Example, Inverse of **2 X 2** matrix, $A = \begin{bmatrix} p & q \\ r & s \end{bmatrix}$

$$A^{-1} = \frac{1}{|A|}\begin{bmatrix} p & q \\ r & s \end{bmatrix} = \frac{1}{|ad-bc|}\begin{bmatrix} s & -q \\ -r & p \end{bmatrix}$$

```
#Printing inverse of a matrix
import numpy as np
x = np.array([[1,2],[3,4]])
y = np.linalg.inv(x)
print(x)
print(y)
print(np.dot(x,y))
```

Solving Linear Equation Problem using Numpy

- A system of Linear Equations can be represented in matrix form using a coefficient matrix, a variable matrix, and a constant matrix.
- Consider the system,

  1a + 1b = 35

  2a + 4b = 94

  Coefficient matrix , $A = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix}$

  Variable Matrix, $X = \begin{bmatrix} a \\ b \end{bmatrix}$

  Constant Matrix, $B = \begin{bmatrix} 35 \\ 94 \end{bmatrix}$

Therefore in short linear equations represented as : **A.X = B**
Variables are calculated by finding product of A-1 and B: **$X = A^{-1} . B$**

```
import numpy as np
#A: Coefficient Matrix
A=[[1,1],[2,4]]
A=np.array(A)
#print(type(A))
print(A)
#B: Constant Matrix
B=[[35],[94]]
B=np.array(B)
#print(type(B))
print(B)
```

```
detA=np.linalg.det(A)
if detA==0:
    print("Solution is not possible!")
else:
    InvA = np.linalg.inv(A)
    C= np.matmul(InvA,B)
    print("Solution: \n",C)
```

Find the solution to the following system of equations.

$$2x + 5y + 2z = -38$$
$$3x - 2y + 4z = 17$$
$$-6x + y - 7z = -12$$

Solving using Numpy:

```
import numpy as np
#A: Coefficient Matrix
A=[[2,5,2],[3,-2,4],[-6,1,-7]]
A=np.array(A)
#print(type(A))
print(A)
#B: Constant Matrix based on Solution
b=[[-38],[17],[-12]]
b=np.array(b)
#print(type(B))
print(b)
detA=np.linalg.det(A)
if detA==0:
    print("Solution is not possible!")
else:
    InvA = np.linalg.inv(A)
    C= np.matmul(InvA,b)
    print("Solution: x = {}, y= {},
z={}".format(C[[0]],C[[1]],C[[2]]))
```

Output:

```
[[ 2  5  2]
 [ 3 -2  4]
 [-6  1 -7]]
[[-38]
 [ 17]
 [-12]]
Solution: x = [[3.]], y= [[-8.]], z=[[-2.]]
```

DATA WRANGLING USING SCIPY

SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. With SciPy, an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems, such as MATLAB, IDL, Octave, R-Lab, and SciLab.

The additional benefit of basing SciPy on Python is that this also makes a powerful programming language available for use in developing sophisticated programs and specialized applications. Scientific applications using SciPy benefit from the development of additional modules in numerous niches of the software landscape by developers across the world. Everything from parallel programming to web and data-base subroutines and classes have been made available to the Python programmer. All of this power is available in addition to the mathematical libraries in SciPy.

- SciPy is built in top of the NumPy
- SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
- Most new Data Science features are available in Scipy rather than Numpy.

You can also install SciPy in Windows via pip

```
Python3 -m pip install --user numpy scipy
```

Install Scipy on Linux

```
sudo apt-get install  python-scipy python-numpy
```

Install SciPy in Mac

```
sudo port install py35-scipy py35-numpy
```

Different subpackages of SciPy

Subpackage	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions

In this chapter, we will cover important sub-packages relevant to us.

File IO (scipy.io)

Scipy, I/O package, has a wide range of functions for work with different files format which are Matlab, Arff, Wave, Matrix Market, IDL, NetCDF, TXT, CSV and binary format.

```
import numpy as np
from scipy import io as sio
#creating an array using Numpy
array = np.ones((4, 4))
#Saving to file
sio.savemat('array.mat', {'ar': array})
#reading from the file
data = sio.loadmat('array.mat', struct_as_record=True)
print("Print the content: \n",data)
#printing only the array part
print("Print the array: \n",data['ar'])
```

Special Package (scipy.special)

SciPy also gives functionality to calculate Permutations and Combinations. In this section, we will only cover sample program. We have covered this topic in detail in our book "*Essential Probability & Statistics for Machine Learning (Implementation using Python) By Swapnil Saurav*".

Permutation and combination are the ways to represent a group of objects by selecting them in a set and forming subsets. It defines the various ways to arrange a certain group of data. When we select the data or objects from a certain group it is said to be permutations whereas the order in which they are represented is called combination.

Permutation Formula

A permutation is the choice of r things from a set of n things without replacement and where the order matters.

$^{n}P_{r} = (n!) / (n-r)!$

Combination Formula

A combination is the choice of r things from a set of n things without replacement and where order does not matter.

$$ {}_nC_r = \binom{n}{r} = \frac{{}_nP_r}{r!} = \frac{n!}{r!(n-r)!} $$

Problem 1: In a group of 6 boys and 4 girls, four children are to be selected. In how many different ways can they be selected such that at least one boy should be there?

Solution: We have 4 options as given below:

- We can select 4 boys: 6C4
- We can select 3 boys and 1 girl: 6C3 × 4C1
- We can select 2 boys and 2 girls: 6C2 × 4C2
- We can select 1 boy and 3 girls: 6C1 × 4C3

```
from scipy.special import comb

sum=0
#find combinations of 5, 2 values using comb(N, k)
#We can select 4 boys: 6C4
com = comb(6, 4, exact = False, repetition=False)
sum+=com
#We can select 3 boys and 1 girl: 6C3 × 4C1
com = comb(6, 3) * comb(4, 1)
sum+=com
#We can select 2 boys and 2 girls: 6C2 × 4C2
com = comb(6, 2) * comb(4,2)
sum+=com
#We can select 1 boy and 3 girls: 6C1 × 4C3
com = comb(6, 1) * comb(4, 3)
sum+=com
print("Total combination possible: ",sum)
```

Output:

Total combination possible: 209.0

The number of combinations of N things taken k at a time. This is often expressed as “N choose k”:

```
scipy.special.comb(N, k, exact=False, repetition=False)
```

Parameters

- Nint, ndarray - Number of things.
- kint, ndarray - Number of elements taken.
- exactbool, optional - If exact is False, then floating point precision is used, otherwise exact long integer is computed.
- repetitionbool, optional - If repetition is True, then the number of combinations with repetition is computed.

Problem 2: Three men have 4 coats, 5 waist coats, and 6 caps. In how many ways can they wear them?

Solution:
Number of ways in which 3 Men can wear 4 coats = 4!/1! = 4! = 24
Number of ways in which 3 Men can wear 5 waist coats = 5 X 4 X 3 = 60
Number of ways in which 3 Men can wear 6 caps = 6 X 5 X 4 = 120

By Fundamental Principle of Multiplication
Arrangement of 4 Coats AND Arrangement of 5 Waist Coats AND Arrangement of 6 Caps
Total number of ways = 24 X 60 X 120 = 172,800

```
from scipy.special import perm

product=1
#find permutation of 4, 3 using perm (N, k) function
#Number of ways in which 3 Men can wear 4 coats
per = perm(4, 3)
product*=per
#Number of ways in which 3 Men can wear 5 waist coats
per = perm(5, 3)
product*=per
#Number of ways in which 3 Men can wear 6 caps
per = perm(6, 3)
product*=per

print("Total Permutation possible: ",product)
```

Output:

Total Permutation possible: 172800.0

Permutations of N things taken k at a time, i.e., k-permutations of N. It's also known as "partial permutations".

```
scipy.special.perm(N, k, exact=False)
```

Parameters

- Nint, ndarray - Number of things.
- kint, ndarray - Number of elements taken.
- exactbool, optional - If exact is False, then floating point precision is used, otherwise exact long integer is computed.

Bessel functions of real order(jv, jn_zeros)

Bessel functions are a family of solutions to Bessel's differential equation with real or complex order alpha:

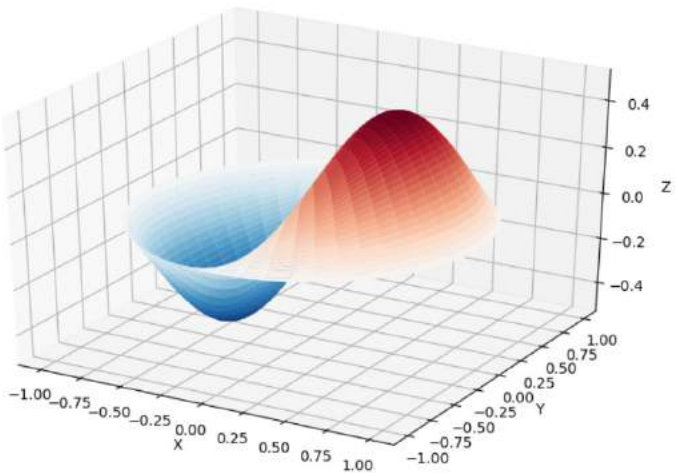
$$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$

Among other uses, these functions arise in wave propagation problems, such as the vibrational modes of a thin drum head. Here is an example of a circular drum head anchored at the edge:

```
from scipy import special
import numpy as np
def drumhead_height(n, k, distance, angle, t):
   kth_zero = special.jn_zeros(n, k)[-1]
   return np.cos(t) * np.cos(n*angle) * special.jn(n,
distance*kth_zero)
theta = np.r_[0:2*np.pi:50j]
radius = np.r_[0:1:50j]
x = np.array([r * np.cos(theta) for r in radius])
```

```
y = np.array([r * np.sin(theta) for r in radius])
z = np.array([drumhead_height(1, 1, r, theta, 0.5) for r in
radius])

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap='RdBu_r',
vmin=-0.5, vmax=0.5)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```

Output



Solving Linear Modelling using Scipy

Linear programming (LP) (also called linear optimization) is a method to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements are represented by linear relationships. Linear programming is a special case of mathematical programming (mathematical optimization).

Linear programming is an extention of Linera algebra we discussed in Numpy. In this section we will solve LP problems with Scipy. You see there are some equations on next page. Here the first equation (to be maximed) is called the Objective function. We either maximize (e.g. Sales, Profit, etc) or minimize (e.g. Cost, Wastage, etc) using this function. We can easily convert a maximization problem into minimization by multiplying by -1 on both the sides. This function is

subjected to some constraints. We can solve by variety of ways. In this section, we will see the graph method, Excel solution and Python Scipy package.

- A **linear function to be maximized**

  e.g. $f(x_1, x_2) = c_1 x_1 + c_2 x_2$
- **Problem constraints** of the following form

  e.g.

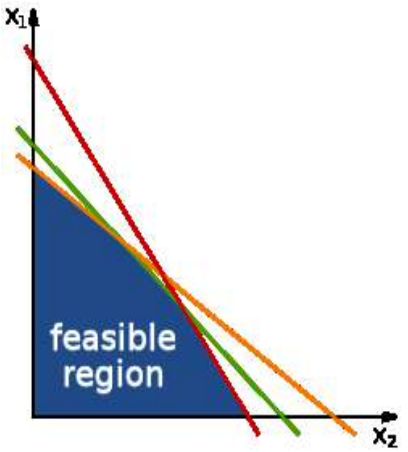
  $$a_{11} x_1 + a_{12} x_2 \leq b_1$$
  $$a_{21} x_1 + a_{22} x_2 \leq b_2$$
  $$a_{31} x_1 + a_{32} x_2 \leq b_3$$
- **Non-negative variables**

  e.g.

  $$x_1 \geq 0$$
  $$x_2 \geq 0$$

The problem is usually expressed in *matrix form*, and then becomes:

$$\max\{\mathbf{c}^{\mathrm{T}}\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b} \wedge \mathbf{x} \geq 0\}$$



Problem Statement:

Let's take a problem to understand the aim and also solve using different methods. Indigo Computers is a manufacturer of Notebooks and Desktops. They make quarterly decisions about their product mix. Indigo Computers would like to know how many of each product to produce in order to **maximize** profit for the quarter. As like any business, Indigo too has some major constraints like:

- Each computer (either notebook or desktop) requires a Processing Chip. Due to tightness in the market, their supplier has allocated only 10,000 such chips to Indigo.
- Each computer requires memory. Memory comes in 16MB chip sets. A notebook computer has 16MB memory installed (so needs 1 chip set) while a desktop computer has 32MB (so requires 2 chip sets). As per the contract negotiation with the suppliers, they have a stock of 15,000 chip sets to use over the next quarter.
- Each computer requires assembly time. Due to tight tolerances, a notebook computer takes more time to assemble: 4 minutes versus 3 minutes for a desktop. There are 25,000 minutes of assembly time available in the next quarter.

Given current market conditions, each notebook computer produced generates Rs 750 profit, and each desktop produces Rs 1000 profit.

There are many questions Indigo Computer might ask. The most obvious are such things as:

- How many of each type computer should Indigo Computer produce in the next quarter? What is the maximum profit Indigo Computer can make?"

Less obvious, but perhaps of more managerial interests are:

- How much should Indigo Computer be willing to pay for an extra memory chip set?

- What is the effect of losing 1,000 minutes of assembly time due to an unexpected machine failure?
- How much profit would we need to make on a 32MB notebook computer to justify its production?

Linear programming gives us a mechanism for answering all of these questions quickly and easily. There are three steps in applying linear programming: modeling, solving, and interpreting.

Modeling a problem using linear programming involves writing it in the language of linear programming. Key to a linear program are the decision variables, objective, and constraints.

Decision Variables: The decision variables represent (unknown) decisions to be made. For this problem, the decision variables are the number of notebooks to produce and the number of desktops to produce. We will represent these unknown values by x_1 and x_2 respectively. To make the numbers more manageable, we will let x_1 be the number of 1000 notebooks produced (so x_1 = 5 means a decision to produce 5000 notebooks) and x_2 be the number of 1000 desktops. Note that a value like the quarterly profit is not (in this model) a decision variable: it is an outcome of decisions x_1 and x_2.

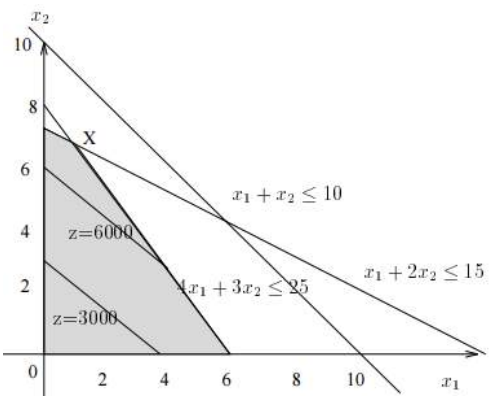
Objective: The objective is to be either minimized or maximized. In this case, our objective is to maximize the function 750x_1 + 1000x_2

Constraints: Here we have four types of constraints: Processing Chips, Memory Sets, Assembly, and Nonnegativity.

- In order to satisfy the limit on the number of chips available, it is necessary that x_1 + x_2 <= 10. If this were not the case (say x_1 = x_2= 6), the decisions would not be implementable (12,000 chips would be required, though we only have 10,000).
- The constraint for memory chip sets is x_1 + 2x_2 <= 15, a linear constraint.
- Our constraint on assembly can be written 4x_1 + 3x_2 <= 25, again a linear constraint.
- Linear constraints x_1 >= 0, x_2 >= 0 to enforce nonnegativity of production.

Final Model: This gives us the complete model of this problem:

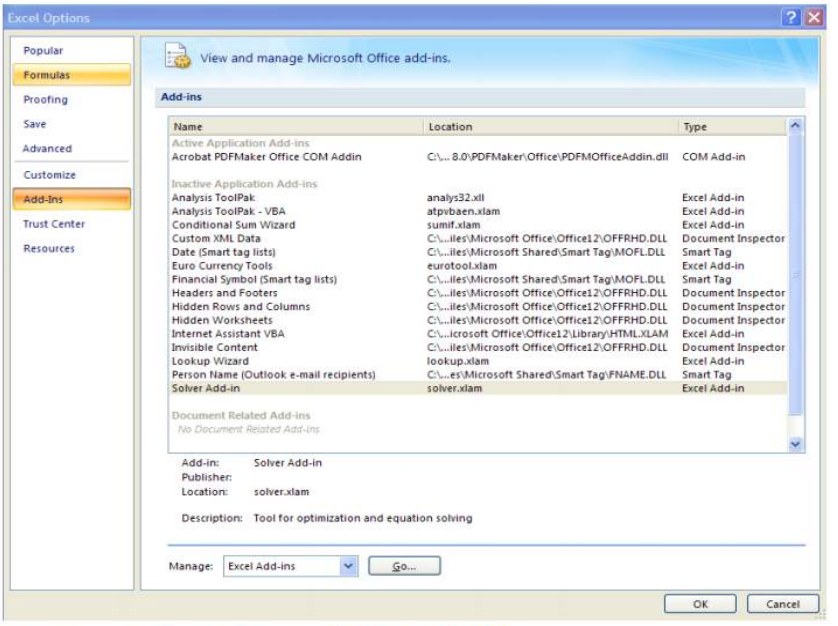
$$
\begin{aligned}
x_1 + x_2 &\leq 10 \\
x_1 + 2x_2 &\leq 15 \\
4x_1 + 3x_2 &\leq 25 \\
x_1 &\geq 0 \\
x_2 &\geq 0
\end{aligned}
$$



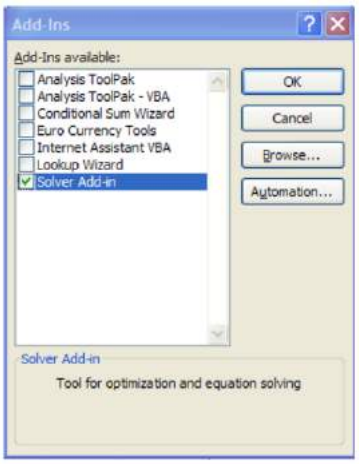
To find the optimal solution, we can include the objective function on this diagram by drawing iso-profit lines: lines along which the profit is the same. Since our goal is to maximize the profit, we can push the isoprofit line out until moving it any further would result in no feasible point (see diagram, z represents profit). Clearly the optimal profit occurs at point X. Note that X is the intersection of the constraints. The solution here is x_1 = 1 and x_2 = 7. The optimal decision is to produce 1,000 notebooks and 7,000 desktops, for a profit of Rs 7,750,000.

Let's solve using Excel first then we will use Numpy to solve.

Install the Solver Add-In



1. In the Microsoft Office button, go to excel options to click Add-ins
2. In the Add-Ins box, select Solver Add-In and click Go...

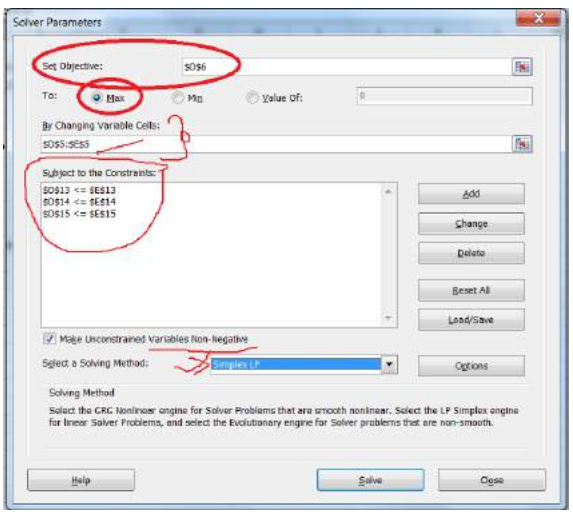


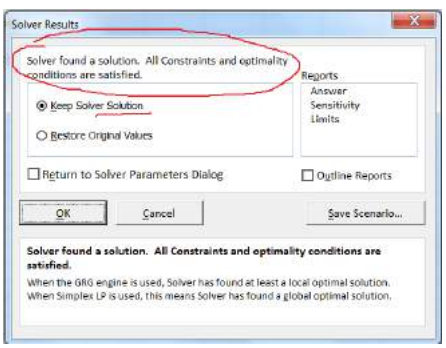
3. In the Add-Ins available box, check the Analysis ToolPak and then OK

	A	B	C	D	E	F	G	H	I
1									
2				Notebook	Desktop				
3			Variables	X1	X2	(Units)			
4			Coefficient	750	1000	(profit per unit)			
5			Solution			(# of units)			
6			Z	=SUMPRODUCT(D4:E4,D5:E5)					
7				SUMPRODUCT(array1, [array2], [array3], [array4], ...)					
8			Constraint 1	1	1	<=	10,000	Processing Chip	
9			Constraint 2	1	2	<=	15,000	Memory	
10			Constraint 3	4	3	<=	25,000	Assembly time	
11									
12				LHS	RHS				
13			Constraint 1	0	10,000	=SUMPRODUCT(D5:E5,D8:E8)			
14			Constraint 2	0	15,000	=SUMPRODUCT(D5:E5,D9:E9)			
15			Constraint 3	0	25,000	=SUMPRODUCT(D5:E5,D10:E10)			
16									

Formulatng the problem:

1. Enter the coefficients of the objective function Z i.e., (750, 1000) in cells D4 and E4. Each unit of Notebook and Desktop will give profit of Rs. 750 and Rs 1,000.
2. Enter the coefficients of the Constraint-1 i.e., (1,1) and RHS value 10,000 in cells D8, E8 and G8 respectively. This is because 1 unit of Processing chip will be consumed by Notebook and Desktop and since the total Processing chip available is 10,000, consumed Processing chips by both should be less than 10,000
3. Enter the coefficients of the Constraint-2 i.e., (1,2) and RHS value 15,000 in cells D9, E9 and G9 respectively. This represents Memory consumption by each unit of Notebook and Desktop.
4. Enter the coefficients of the Constraint-3 i.e., (4,3) and RHS value 25,000 in cells D10, E10 and G10 respectively. This represents Assembly time constraint by each unit of Notebook and Desktop.
5. Solution (C5) represents the number of units that are needed to be produced to meet maximum profit (D5 & E5). We have to find these value so for now leave them blank.
6. Total Profit (Z) is the objective function and it will calculated as: D4*D5 + E4*E5, which can be represented as SUMPRODUCT in excel as shown above. D6 will hold the value and we need to maximize this number using the Solver.





7. Now we represent the values of the constraints to get maximum profit. These values are also calculated as the SUMPRODUCT as shown above under LHS (Left Hand Side) header. The Right Hand Side values (RHS) will be the maximum values that these constraints can take.

Implementing the Solution

1. Start the Solver
2. Set the Objective which is D6 in our example
3. Objective needs to be maximized since we are finding the maximum profit that's possible within given constraints
4. Enter the three constraints. The LHS value needs to be less than or equal to RHS values hence we have entered LHS and RHS cell references
5. We will be solving using the Simplex Method hence select the Simplex method. Simplex method is the most popular method to solve LP Problems. You can refer additional material to learn more about the method. Now click on Solve button.

	Notebook	Desktop			
Variables	X1	X2	(Units)		
Coefficient	750	1000	(profit per unit)		
Solution	1000	7000	(# of units)		
Z	7750000		(Total Profit)		
Constraint 1	1	1	<=	10,000	Processing Chip
Constraint 2	1	2	<=	15,000	Memory
Constraint 3	4	3	<=	25,000	Assembly time
	LHS	RHS			
Constraint 1	8000	10,000			
Constraint 2	15000	15,000			
Constraint 3	25000	25,000			

Now, let's solve using Scipy package: Relook at the equations:

$$\begin{aligned} x_1 + x_2 &\leq 10 \\ x_1 + 2x_2 &\leq 15 \\ 4x_1 + 3x_2 &\leq 25 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Now, we need to form the Matrices:

$$\begin{pmatrix} [[1\ 1] \\ [1\ 2] \\ [4\ 3] \\ [1\ 0] \\ [0\ 1]] \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} [[10] \\ [15] \\ [25] \\ [\ 0] \\ [\ 0]] \end{pmatrix}$$

Using the Optimize Module in SciPy
When you need to optimize the input parameters for a function, scipy.optimize contains a number of useful methods for optimizing different kinds of functions:

- minimize_scalar() and minimize() to minimize a function of one variable and many variables, respectively
- curve_fit() to fit a function to a set of data
- root_scalar() and root() to find the zeros of a function of one variable and many variables, respectively
- linprog() to minimize a linear objective function with linear inequality and equality constraints

scipy.optimize also includes the more general minimize(), which we will use for our problem. This function can handle multivariate inputs and outputs and has more complicated optimization algorithms to be able to handle this. In addition, minimize() can handle constraints on the solution to your problem. You can specify three types of constraints:

- LinearConstraint: The solution is constrained by taking the inner product of the solution x values with a user-input array and comparing the result to a lower and upper bound.
- NonlinearConstraint: The solution is constrained by applying a user-supplied function to the solution x values and comparing the return value with a lower and upper bound.
- Bounds: The solution x values are constrained to lie between a lower and upper bound.

Our objective function is to maximize but the function available is minimize so, we need to change the sign of our objective function.

Original function: maximize (750x_1 + 1000x_2)
Now, minimize (750x_1 +1000x_2)*(-1) = -750x_1 – 1000x_2 – multiplied by -1

```
import numpy as np
from scipy.optimize import minimize, LinearConstraint
from scipy.optimize import linprog

#maximize profit - starting with some random values for variables
x_1 = 10 #the number of 1000 notebooks
x_2 = 10 #x_2 be the number of 1000 desktops
profit_per_nb = 750
profit_per_dt = 1000
profit = (profit_per_nb * x_1 + profit_per_dt * x_2) *1000 #in
'000
#to minimize
##  -profit =  -profit_per_nb * x_1 - profit_per_dt * x_2
obj = [-profit_per_nb, - profit_per_dt]  #Coefficient of x and y
variables

lhs_constraints_ineq = [[1,1], # equation 1
                        [1,2],      # equation 2
```

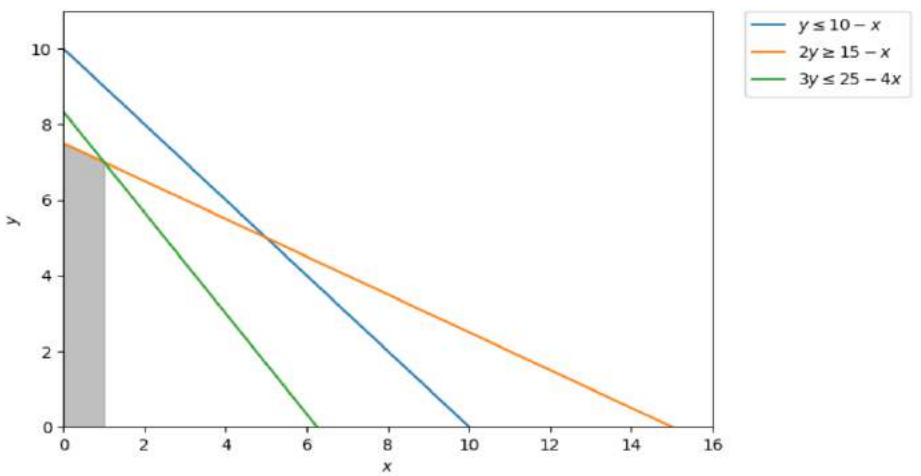
```
                    [4,3]]      # equation 3
rhs_value_ineq = [10,      # equation 1
                  15,      # equation 2
                  25]      # equation 3

#lhs_eq & rhs_eq if we have any equality constraints,
#In our example we do not have hence ignoring

#Below mentioning bound condition -
# x_1 and x_2 need to be positive integer numbers hence:
bnd = [(0, float("inf")),  # Bounds of x_1
       (0, float("inf"))]  # Bounds of x_2

opt = linprog(c=obj, A_ub=lhs_constraints_ineq,
b_ub=rhs_value_ineq,
    bounds=bnd,     #you can add equality contraints to: A_eq and
b_eq
    method="revised simplex")
print(opt)
if opt.success:
    print("Total number of Notebooks that should be manufactured:
", opt.x[0]*1000)
    print("Total number of Desktops that should be manufactured:
", opt.x[1]*1000)
    print("Maximum profit that can be realized: ", -
1*opt.fun*1000) #Time to change the sign

else:
    print("There is no solution to the problem!")

#Plot the graphs now
import matplotlib.pyplot as plt

# Construct lines
# x_1 > 0 & x_2 >0
x1 = np.linspace(0, 20, 8000)
# x_2 > 0
y1 = (x1*0)
# Eq1: x1+x2 <=10
y2 = 10-x1
# Eq2: x1+2x2 <=15
y3 = (15-x1)/2.0
# Eq3: 4x1+3x2 <=25
y4 = (25-4*x1)/3.0

# Make plot
#plt.plot(x1, y1, label=r'$y\geq0$')
plt.plot(x1, y2, label=r'$y\leq10-x$')
```

```
plt.plot(x1, y3, label=r'$2y\geq 15 - x$')
plt.plot(x1, y4, label=r'$3y\leq 25-4x$')
plt.xlim((0, 16))
plt.ylim((0, 11))
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')

# Fill feasible region
y5 = np.minimum(y2, y4)
y6 = np.minimum(y2, y3)
plt.fill_between(x1, y1, y6, where=y5>y6, color='grey', alpha=0.5)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```

Output

```
     con: array([], dtype=float64)
     fun: -7750.0
 message: 'Optimization terminated successfully.'
     nit: 2
   slack: array([2., 0., 0.])
  status: 0
 success: True
       x: array([1., 7.])
Total number of Notebooks that should be manufactured:  1000.0
Total number of Desktops that should be manufactured:  7000.0
Maximum profit that can be realized:  7750000.0
```

The result tells you that the maximal profit is 7,750,000 and corresponds to x_1 = 1000 (# of Notebooks) and x_2 = 7000 (# of Desktops).



PuLP Package: PuLP has a more convenient linear programming API than SciPy. You don't have to mathematically modify your problem or use vectors and matrices. Everything is cleaner and less prone to errors. Its left for the readers to practice this Python package.

DATA WRANGLING USING PANDAS

Pandas package is for data extraction and preparation. Pandas is a very popular library that provides high-level data structures which are simple to use as well as intuitive. It has many inbuilt methods for grouping, combining data and filtering as well as performing time series analysis. Pandas can easily fetch data from different sources like SQL databases, CSV, Excel, JSON files and manipulate the data to perform operations on it.

NOTE: ALL THE DATASET USED IN THIS BOOK CAN BE DOWNLOADED FROM GITHUB LOCATION: HTTPS://GITHUB.COM/SWAPNILSAURAV/THEATOZOFDATASCIENCEBOOK

Pandas deals with the following three data structures –

- Series
- DataFrame

Data Structure	Dimensions	Descriptions
Series	1	1D labeled homogeneous array, size immutable
DataFrame	2	General 2D labeled, size-mutable tabular structure

These data structures are built on top of Numpy array, which means they are fast. The best way to think of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure. For example, DataFrame is a container of Series. Panel was a container of DataFrame for 3D labeled data but now it has been removed. The recommended way to represent 3-D data are with a MultiIndex on a DataFrame.

SERIES

Series is a one-dimensional array like structure with homogeneous data. For example, the ollowing series is a collection of integers:

44	**33**	**66**	**88**	**77**	**99**	**55**

PANDAS.SERIES

pandas.Series: A pandas Series can be created using the following constructor –

pandas.Series(data, index, dtype, copy)

The parameters of the constructor are as follows –

S. No.	Parameter & Description
1	data: data takes various forms like ndarray, list, constants
2	index: Index values must be unique and hashable, same length as data. Default np.arrange(n) if no index is passed
3	dtype: dtype is for data type. If None, data type will be Object
4	copy: Copy data. Default False

Example: Creating empty series

```
import pandas as pd
s = pd.Series(dtype= "float64")
print(s)
```

Example: Create a Series from ndarray

```
#Example 2: Create a Series from ndarray
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print(s)
```

We did not pass any index, so by default, it assigned the indexes ranging from 0 to len(data)-1

Example

```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[30,33,36,39])
print(s)
```

We passed the index values here. Now we can see the customized indexed values in the output.

Example: More examples of creating Series:

```
import pandas as pd
import numpy as np
#Create a Series from dict
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print(s)
#Create a Series from Scalar
s = pd.Series(5, index=[0, 1, 2, 3])
print(s)
```

Note: If data is a scalar value, an index must be provided. The value will be repeated to match the length of index.

Example: Accessing Data from Series with Position

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
# Retrieve the first element, which is stored at zeroth position.
print(s[0])
#retrieve the first four element
print(s[:4])
#retrieve the last four element
print(s[-4:])
```

Refer the chapter on Strings to understand more on indexing. Strings are part of Basic Programming concept and has been covered in the book "Learn and Practice Python Programming by Swapnil Saurav".

Example: Retrieve Data Using Label (Index)

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
#retrieve a single element
print(s['a'])
#retrieve multiple elements
print(s[['a','c','d']])
```

DATAFRAME

DataFrame is a two-dimensional array with heterogeneous data. For example,

ROLLNO	NAME	BRANCH	PHONE	EMAIL
45	Ajay	CSE	9945689651	ajay.cse45@example.org
46	Sachin	ECE	7896523459	sachin.ece46@ example.org
48	Kapil	CE	9009876456	kapil.ce48@ example.org
49	Saurabh	CHE	9087136745	saurabh.che49@ example.org

The data is represented in rows and columns. Each column represents an attribute and each row represents a person (specific record).

PANDAS.DATAFRAME

pandas.DataFrame: A pandas DataFrame can be created using the following constructor –

pandas.DataFrame(data, index, columns, dtype, copy)

The parameters of the constructor are as follows:

Parameter	Description
data	data takes various forms like ndarray, series, map, lists, dict, constants and also

Parameter	Description
	another DataFrame.
index	For the row labels, the Index to be used for the resulting frame is Optional Default np.arrange(n) if no index is passed.
columns	For column labels, the optional default syntax is - np.arrange(n). This is only true if no index is passed.
dtype	Data type of each column.
copy	This command (or whatever it is) is used for copying of data, if the default is False.

Example: Working with DataFrames

```
#import the pandas library and aliasing as pd
import pandas as pd
#Create an Empty DataFrame
df = pd.DataFrame()
print(df)

#Create a DataFrame from Lists
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print(df)
#Create a DataFrame from 2D Lists
data = [['Ajay',40],['Baran',52],['Chandan',33]]
df = pd.DataFrame(data,columns=['Name','Avg'])
print(df)
# with float value
data = [['Ajay',40],['Baran',52],['Chandan',33]]
df = pd.DataFrame(data,columns=['Name','Avg'],dtype=float)
print(df)
#Create a DataFrame from Dict of ndarrays / Lists
data = {'Name':['Ram', 'Jad', 'Mahi',
'Sachin'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print(df)

#Create a DataFrame from List of Dicts
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print(df)
#Note: NaN (Not a Number) is appended in missing areas.
#by passing a list of dictionaries and the row indices
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data, index=['first', 'second'])
print(df)

#with a list of dictionaries, row indices, and column indices.
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
```

```
#With two column indices, values same as dictionary keys
df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a',
'b'])

#With two column indices with one index with other name
df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a',
'b1'])
print(df1)
print(df2)
#Note: df2 DataFrame is created with a column index other than the
dictionary key; thus, appended the NaN's in place. Whereas, df1 is
created with column indices same as dictionary keys, so NaN's
appended.

#Create a DataFrame from Dict of Series
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print(df)
```

CREATING DATAFRAMES FROM VARIOUS DATA

Let's cover some additional topics like creating a DataFrame object from Numpy arrays, from list of tuples, from dictionary, etc.

```
import pandas as pd
import numpy as np

#Form a Dataframe
data = {
    'apples': [25000, 15000, 18000, 21000],
    'oranges': [80000, 34000, 12000, 12000]
}
#pass it to the pandas DataFrame constructor:
production = pd.DataFrame(data)
#Each (key, value) item in data corresponds to a column in the
resulting DataFrame.
print(production)

#The Index of this DataFrame was given to us on creation as the
numbers 0-3,
# but we could also create our own when we initialize the
DataFrame.
purchases = pd.DataFrame(data, index=['Jan', 'Feb', 'Mar', 'Apr'])
print(production)
#So now we could locate a production details by using month:
```

```
print(purchases.loc['Mar'])

#Reading from a csv file
df2 = pd.read_csv('employee_file.csv')
#CSVs don't have indexes like our DataFrames, so
#we will designate the index_col =0
df = pd.read_csv('employee_file.csv', index_col=0)
print(df2)

#Reading data from JSON
df3 = pd.read_json('person.json')
print(df3)

#Connecting to Database (sqllite)
import sqlite3
con = sqlite3.connect("database.db")
df4 = pd.read_sql_query("SELECT * FROM Employee", con)
print(df4)
#Create a DataFrame from a dictionary, containing two columns
df = pd.DataFrame({'RollNo': [101, 102, 103], 'Names': ['Rahul',
'Rohit', 'Ramesh']})
print(df)
#Pandas orders columns alphabetically as dict are not ordered.
# To specify the order, use the columns parameter.
df = pd.DataFrame({'RollNo': [101, 102, 103], 'Names': ['Rahul',
'Rohit', 'Ramesh']},
 columns=['Names', 'RollNo'])
print(df)
```

Slicing and Dicing in a Dataframe

In the programs put in this section, we will see how to read a subset of data from a dataframe and also how to modify the data.

```
import pandas as pd
import numpy as np

#Output Names columns come before RollNo
#Create a DataFrame of random numbers:
# Set the seed for a reproducible sample
np.random.seed(0)
df = pd.DataFrame(np.random.randn(4, 3), columns=list('XYZ'))
print(df)
#Create a DataFrame with integers:
df =
pd.DataFrame(np.arange(42).reshape(7,6),columns=list('ABCDEF'))
print(df)
```

```
# iloc[row slicing, column slicing]
#iloc has 2 parts before , represents the rows and
#after comma represents the columns
#Slicing is similar to String
print("1: \n",df.iloc[0:3, 1:4])

# Select all columns for rows of index values 0 and 2
print("2: \n",df.loc[[0, 2], :])

# Select first row of columns B,C,D
print("3: \n",df.loc[0, ['B', 'C', 'D']])

# Select first, third and fifth rows of all columns
print("4: \n",df.loc[[0, 2, 4], :])

#Include nan across second row all columns
 # in column 0, set elements with indices 1,2 ... to NaN
print("NaN:")
df.iloc[[1,2],0] = np.nan
# in column 1, set elements with indices 0,4, to np.NaT (Not a
Time)
df.iloc[[0,4],1] = pd.NaT
# in column 2, set elements with index from 0 to 3 to 'nan'
df.iloc[0:3,2] = 'nan'
# in column 5, set all elements to None
df.iloc[:,5] = None
# in row 5, set all elements to NaN
df.iloc[5,:] = np.nan
print(df)
```

Output is shown on next page: shown in 3 columns for better readability

```
          X         Y         Z
0  1.764052  0.400157  0.978738
1  2.240893  1.867558 -0.977278
2  0.950088 -0.151357 -0.103219
3  0.410599  0.144044  1.454274
    A   B   C   D   E   F
0   0   1   2   3   4   5
1   6   7   8   9  10  11
2  12  13  14  15  16  17
3  18  19  20  21  22  23
4  24  25  26  27  28  29
5  30  31  32  33  34  35
6  36  37  38  39  40  41
```

```
1:
    B   C   D
0   1   2   3
1   7   8   9
2  13  14  15
2:
    A   B   C   D   E   F
0   0   1   2   3   4   5
2  12  13  14  15  16  17
3:
 B    1
C    2
D    3
Name: 0, dtype: int32
4:
    A   B   C   D   E   F
0   0   1   2   3   4   5
2  12  13  14  15  16  17
4  24  25  26  27  28  29
```

```
NaN:
     A    B    C     D     E     F
0  0.0  NaT  nan   3.0   4.0  None
1  NaN    7  nan   9.0  10.0  None
2  NaN   13  nan  15.0  16.0  None
3 18.0   19   20  21.0  22.0  None
4 24.0  NaT   26  27.0  28.0  None
5  NaN  NaN  NaN   NaN   NaN   NaN
6 36.0   37   38  39.0  40.0  None
```

Creating Dataframe with Dates

```
import pandas as pd
import numpy as np
np.random.seed(0)
# create an array of 5 dates starting at '2020-08-15', one per
minute
rng = pd.date_range('2020-08-15', periods=5, freq='T')
df = pd.DataFrame({ 'Date': rng, 'Val': np.random.randn(len(rng))
})
print (df)
# create an array of 5 dates starting at '2020-08-15', one per day
rng = pd.date_range('2020-08-15', periods=5, freq='D')
df = pd.DataFrame({ 'Date': rng, 'Val' :
np.random.randn(len(rng))})
print (df)
# create an array of 5 dates starting at '2020-08-15', one every 3
years
rng = pd.date_range('2020-08-15', periods=5, freq='3A')
df = pd.DataFrame({ 'Date': rng, 'Val' :
np.random.randn(len(rng))})
print (df)
np.random.seed(0)
rng = pd.date_range('2020-08-15', periods=5, freq='T')
df = pd.DataFrame({ 'Val' : np.random.randn(len(rng)) },
index=rng)
```

Output (presented in 3 columns)

	Date	Val
0	2020-08-15 00:00:00	1.764052
1	2020-08-15 00:01:00	0.400157
2	2020-08-15 00:02:00	0.978738
3	2020-08-15 00:03:00	2.240893
4	2020-08-15 00:04:00	1.867558

	Date	Val
0	2020-08-15	-0.977278
1	2020-08-16	0.950088
2	2020-08-17	-0.151357
3	2020-08-18	-0.103219
4	2020-08-19	0.410599

	Date	Val
0	2020-12-31	0.144044
1	2023-12-31	1.454274
2	2026-12-31	0.761038
3	2029-12-31	0.121675
4	2032-12-31	0.443863

COLUMN SELECTION, ADDITION, AND DELETION IN A DATAFRAME

Example: Let us now understand column selection and column addition through examples.

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
#Column value selection
```

```
print( df ['one'])

# Adding a new column to an existing DataFrame object with column
label by passing new series

print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print(df)

print ("Adding a new column using the existing columns in
DataFrame:")
df['four']=df['one']+df['three']

print(df)
```

Explanation: 4th column is added by adding of the values from column "one" and "three".

Example: Let us now understand column deletion from a data frame through examples.

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
     'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)
print ("Our dataframe is:")
print(df)

# using del function
print ("Deleting the first column using DEL function:")
del df['one']
print(df)

# using pop function
print ("Deleting another column using POP function:")
df.pop('two')
print(df)
```

ROW SELECTION, ADDITION, AND DELETION IN A DATAFRAME

Example: Let us now understand row selection and row addition through examples.

```
import pandas as pd

#Appending a row by a single column value:
df3 = pd.DataFrame(columns=['Name', 'Location', 'Phone'])
df3.loc[0, 'Name'] = "Javagal"
```

```
print(df3)

#Appending a row, given list of values:
df3.loc[1] = ["Anil", "Bangalore", 12345]
print(df3)

#Appending a row given a dictionary:
df3.loc[2] = {'Name': "Romesh", 'Location': "Mumbai", 'Phone':
12349}
print(df3)
#Overwrite
df3.loc[2] = {'Name': "Ramesh", 'Location': "Mumbai", 'Phone':
12349}
print(df3)

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print(df)
#Rows can be selected by passing row label to a loc function
print(df.loc['b'])

#Slice Rows : Multiple rows can be selected using ' : ' operator.
print(df[2:4])

print("Add new rows to a DataFrame using the append function")
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
df1 = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])

df1 = df1.append(df2, sort=True)
print(df1)
```

In this above example, two rows were dropped because those two contain the same label 0.

Boolean indexing of dataframes

What if the index is Boolean? Let's look at the some examples to play with Boolean index using .loc and .iloc

```
import pandas as pd
#Index represents if its a rainbow color
df = pd.DataFrame({"color": ['red', 'blue', 'orange', 'green',
                             'yellow','brown','black','white',
                             'indigo','violet','Pink','Purple'],
                   "name": ["Rose", "Sky","Orange","Tree",
                            "Ball","Bread","Bird","Milk",
                            "Car","Book","Dress","Cake"]},
 index=[True,True,True,True,
```

```
        True,False, False, False,
        True, True, False, False])

print("Dataframe: \n",df)
print("Colors of Rainbow: \n",df.loc[True])
print("Non Rainbow Colors:\n",df.loc[False])
print("First Element:\n",df.iloc[0])

#Accessing values
sample_size = df['name'] == 'Rose'
print("Details of Data with Name as Rose: \n",df[sample_size])
```

Output: For better visibility, output is broken into 3 columns

```
Dataframe:
         color    name
True       red    Rose
True      blue     Sky
True    orange  Orange
True     green    Tree
True    yellow    Ball
False    brown   Bread
False    black    Bird
False    white    Milk
True    indigo     Car
True    violet    Book
False     Pink   Dress
False   Purple    Cake
Colors of Rainbow:
         color    name
True       red    Rose
True      blue     Sky
True    orange  Orange
True     green    Tree
True    yellow    Ball
True    indigo     Car
True    violet    Book
Non Rainbow Colors:
          color    name
False    brown   Bread
False    black    Bird
False    white    Milk
False     Pink   Dress
False   Purple    Cake
First Element:
 color      red
name      Rose
Name: True, dtype: object
Details of Data with Name as Rose:
      color  name
True    red  Rose
```

Query data using Pandas

We can use the syntax below when querying data by criteria from a DataFrame. Experiment with selecting various subsets of the "Playerss" data.

- Equals: ==
- Not equals: !=
- Greater than, less than: > or <
- Greater than or equal to >=
- Less than or equal to <=

```
import pandas as pd

players = {"name": pd.Series(["Sachin", "Anil", "Zaheer",
"Sourav"]),
           "primary_type":
pd.Series(["Batsman","Bowler","Bowler","Batsman"]),
           "runs": pd.Series([25000,8000,7000,19000]),
           "wickets": pd.Series([40,700,600,150])}
players_df = pd.DataFrame(players)
players_df = players_df.set_index(players_df["name"])
print ("Our dataframe is:")
print(players_df)
#1. Checking the value
```

```
print("\n1. Get the data of the players in the dataframe:")
print(players_df[players_df['name'].isin(["Sachin", "Kaif",
"Zaheer"])])
#1. Explanation: since Kaif is not in the main data,
# his records are not displayed

#2. Average wickets taken by the players
print("\n2. Average wickets taken: ",end=" ")
print(players_df['wickets'].mean())

#3. Average of all numeric columns based on type of the players
print("\n3. Average value for Player type:\n")
print(players_df.groupby('primary_type').mean())

#4. Get the sums of the runs scored by all the players
print("\n4. Total runs scored:",end=" ")
print(players_df["runs"].sum())
```

Output: set in 3 columns

```
Our dataframe is:
         name primary_type   runs  wickets
name
Sachin  Sachin      Batsman  25000       40
Anil      Anil       Bowler   8000      700
Zaheer  Zaheer       Bowler   7000      600
Sourav  Sourav      Batsman  19000      150

1. Get the data of the players in the dataframe:
         name primary_type   runs  wickets
name
Sachin  Sachin      Batsman  25000       40
Zaheer  Zaheer       Bowler   7000      600

2. Average wickets taken:  372.5

3. Average value for Player type:

                runs  wickets
primary_type
Batsman        22000       95
Bowler          7500      650

4. Total runs scored: 59000
```

DATA MANIPULATION USING DROP()

We can use Drop function of Pandas to drop/delete Rows And Columns In pandas Dataframe. Let's taken an example of following dataframe:

```
import pandas as pd
data1={
    "Name":
["Joy","Pancham","Sabuj","Vivek","Shinjith","Vinay","Priyvrat"],
    "Marks": [95,89,65,78,99,78,87],
    "Branch": ["CSE","ECE","TEL","MECH","CIVIL","EEE","ROBOTICS"]
}
df1 = pd.DataFrame(data1)
#Let it use Default Index of 0,1,...
print("Dataframe 1 with default Index: \n",df1)
df2 = pd.DataFrame(data1, index = [101,102,103,104,105,106,107])
#Let it use Default Index of 0,1,...
print("Dataframe 2 with Roll No. as Index: \n",df2)
```

Output:

```
Dataframe 1 with default Index:
        Name  Marks    Branch
0       Joy     95       CSE
1   Pancham     89       ECE
2     Sabuj     65       TEL
3     Vivek     78      MECH
4  Shinjith     99     CIVIL
5     Vinay     78       EEE
6  Priyvrat     87  ROBOTICS
Dataframe 2 with Roll No. as Index:
          Name  Marks    Branch
101       Joy     95       CSE
102   Pancham     89       ECE
103     Sabuj     65       TEL
104     Vivek     78      MECH
105  Shinjith     99     CIVIL
106     Vinay     78       EEE
107  Priyvrat     87  ROBOTICS
```

Drop an observation (row)

Axis is variable in Pandas which decides if we want to work with Rows or Columns. Axis = 0 indicate it's a row and axis is 1 for columns. Default value of axis is 0.
Drop rows with index 1 and 2 in firstdataframe and 105 and 107 from the second dataframe: so the output should look like:

```
After Drop Dataframe 1 with default Index:
        Name  Marks    Branch
0       Joy     95       CSE
3     Vivek     78      MECH
4  Shinjith     99     CIVIL
5     Vinay     78       EEE
6  Priyvrat     87  ROBOTICS
After Drop Dataframe 2 with Roll No. as Index:
         Name  Marks Branch
101      Joy     95    CSE
102  Pancham     89    ECE
103    Sabuj     65    TEL
104    Vivek     78   MECH
106    Vinay     78    EEE
```

Python Code:

```
#Deleting a row:
df1 = df1.drop([1,2],axis=0)
print("After Drop Dataframe 1 with default Index: \n",df1)
#We can ignore axis value as default value of axis
#is zero anyway
df2 = df2.drop([105,107])
print("After Drop Dataframe 2 with Roll No. as Index: \n",df2)
```

Drop a variable (column)

Note: axis=1 denotes that we are referring to a column, not a row
From dataframe delete column Branch. Output should look something like:

```
After Drop Dataframe 1 with default Index:
        Name  Marks
0        Joy     95
3      Vivek     78
4   Shinjith     99
5      Vinay     78
6   Priyvrat     87
```

Program:

```
#Deleting a variable (column)
#Delete column Branch from df1
df1 = df1.drop("Branch", axis=1)
print("After Drop Dataframe 1 with default Index: \n",df1)
```

More Programs:

```
#Delete row from DF2 where name is Priyvrat
df3 = df2[df2.Name != 'Priyvrat']
#Name= N is cap, Python is case sensitive
print("DF2 after dropping name Priyvrat: \n", df3)

#Drop a row by row number (in this case, row 4)
#Note that Pandas uses zero based numbering,
# so 0 is the first row, 1 is the second row, etc
df4 = df1.drop(df1.index[3])
print("DF1 after dropping index 3: \n", df4)

#dropping relative to the end of the DF.
#Drop second last row in df1
df5 = df1.drop(df1.index[-2])
print("DF1 after dropping second last index: \n", df5)
```

Output:

```
DF2 after dropping name Priyvrat:
          Name  Marks Branch
101        Joy     95    CSE
102    Pancham     89    ECE
103      Sabuj     65    TEL
104      Vivek     78   MECH
106      Vinay     78    EEE
DF1 after dropping index 3:
         Name  Marks
0         Joy     95
3       Vivek     78
4    Shinjith     99
6    Priyvrat     87
DF1 after dropping second last index:
         Name  Marks
0         Joy     95
3       Vivek     78
4    Shinjith     99
6    Priyvrat     87
```

MERGE THE DATASET

The Pandas merge() command takes the left and right dataframes, matches rows based on the "on" columns, and performs different types of merges – left, right, etc.

PANDAS.MERGE()

Dataframe merge can be done using –

```
pandas.merge(left_data, right_data, left_on, right_on, how)
```

The parameters of the constructor are as follows:

Parameter	Description
left_data	"left" dataframe
right_data	"right" dataframe
left_on	left_on specifies merge column names(s) in left dataframe
right_on	right_on specifies merge column names(s) in right dataframe
how	how can be one of: left, right, inner, outer

Example 11: Merge dataframes

In this example, we have taken 3 dataframes which can be downloaded from the github.

- user_usage.csv – A first dataset containing users monthly mobile usage statistics

- user_device.csv – A second dataset containing details of an individual "use" of the system, with dates and device information.
- android_devices.csv – A third dataset with device and manufacturer data, which lists all Android devices and their model code, obtained from Google.

Step 1: Read the csv files

```
import pandas as pd
#Read the 3 csv files
d1 =
pd.read_csv(r'https://raw.githubusercontent.com/swapnilsaurav/Dataset/master/user_usage.csv')
#showing monthly mobile usage statistics for a subset of users.
print(d1.head(5))
d2 =
pd.read_csv(r'https://raw.githubusercontent.com/swapnilsaurav/Dataset/master/user_device.csv')
#giving the device and OS version for individual
print(d2.head(5))
d3 =
pd.read_csv(r'https://raw.githubusercontent.com/swapnilsaurav/Dataset/master/android_devices.csv')
#containing all Android devices with manufacturer and model
details
print(d3.head(5))
```

There are linking attributes between the sample datasets that are important to note – "use_id" is shared between the user_usage and user_device, and the "device" column of user_device and "Model" column of the devices dataset contain common codes. We want to form a single dataframe with columns for user usage figures (calls per month, sms per month etc) and also columns with device information (model, manufacturer, etc). We will need to "merge" (or "join") our sample datasets together into one single dataset for analysis.

Let's see how we can correctly add the "device" and "platform" columns to the user_usage dataframe using the Pandas Merge command.

```
#add the "device" and "platform" columns to the user_usage
dataframe
result = pd.merge(d1,
                  d2[['use_id', 'platform', 'device']],
                  on='use_id')
result.head()
```

Merging requires a "left" dataset, a "right" dataset, and a common column to merge "on". Pandas merge() defaults to an "inner" merge operation. Let's now analyze the return dataframes size. First, let's look at the sizes or shapes of our inputs and outputs to the merge command:

```
#Analyze the dimensions of the dataframes
print("user_usage Dimensions: ",d1.shape)
print("user_device Dimensions: ", d2[['use_id', 'platform',
'device']].shape)
print("result Dimensions: ",result.shape)
```

Output:

```
user_usage Dimensions:  (240, 4)
user_device Dimensions:  (272, 3)
result Dimensions:  (159, 6)
```

Why is the result a different size to both the original dataframes?

An inner merge, (or inner join) keeps only the common values in both the left and right dataframes for the result. In our example above, only the rows that contain use_id values that are common between user_usage and user_device remain in the result dataset. We can validate this by looking at how many values are common:

```
#how many values are common
print(d1['use_id'].isin(d2['use_id']).value_counts())
```

Output:

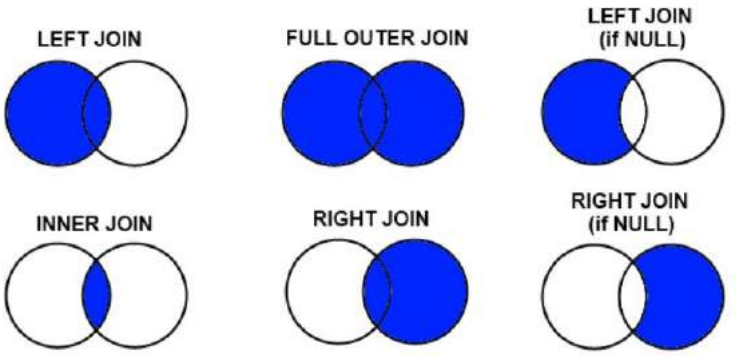
```
True     159
False     81
Name: use_id, dtype: int64
```

This explains why only 159 rows are in the result dataframe.

Other Merge type

Apart from Inner Merge/Inner Join, other types of Merge are:

- Left Merge / Left outer join – Keep every row in the left dataframe. For missing values of the "on" variable in the right dataframe, add empty / NaN values in the result.
- Right Merge / Right outer join –Keep every row in the right dataframe. For missing values of the "on" variable in the left column, add empty / NaN values in the result.
- Outer Merge / Full outer join – A full outer join returns all the rows from the left dataframe, all the rows from the right dataframe, and matches up rows where possible, with NaNs elsewhere.



Merge/Join types as used in Pandas, R, SQL, and other data-orientated languages and libraries. Source: Stack Overflow.

You can change the merge to a left-merge with the "how" parameter to your merge command.

```
#You can change the merge to a left-merge with the "how" parameter
to your merge command.
result = pd.merge(d1,
                  d2[['use_id', 'platform', 'device']],
                  on='use_id',
                  how='left',
                 indicator =True)
result.head()
print("user_usage Dimensions: ",d1.shape)
print("result Dimensions: ",result.shape)
print("There are {} missing values in the
result.".format(result['device'].isnull().sum()))
```

Output:

```
user_usage Dimensions:  (240, 4)
result Dimensions:  (240, 7)
There are 81 missing values in the result.
```

Indicator: To assist with the identification of where rows originate from, Pandas provides an "indicator" parameter that can be used with the merge function which creates an additional column called "_merge" in the output that labels the original source for each row.

Practice Problems

1. Extend the examples given in this section with right merge / right join and outer merge / full outer join

Using left_on and right_on to merge with different column names

Now let's add the third dataframe. We will redo the first merge to get back to inner merge and then merge devices dataframe.

```
# Add the platform and device to the user usage
result = pd.merge(d1,
                  d2[['use_id', 'platform', 'device']],
                  on='use_id',
                  how='left')

# Merge based on the "device" column in result, match the "Model"
column in devices (d3).
d3.rename(columns={"Retail Branding": "manufacturer"},
inplace=True)
result = pd.merge(result,
                   d3[['manufacturer', 'Model']],
                   left_on='device',
                   right_on='Model',
                   how='left')
print(result.head())
```

The columns used in a merge operator do not need to be named the same in both the left and right dataframe. In the second merge above, note that the device ID is called “device” in the left dataframe, and called “Model” in the right dataframe. Different column names are specified for merges in Pandas using the “left_on” and “right_on” parameters, instead of using only the “on” parameter.

Calculating statistics based on device

We can use the data aggregation functionality of Pandas to quickly work out the mean usage for users based on device manufacturer.

```
#Calculating statistics based on device
print(result.groupby("manufacturer").agg({
        "outgoing_mins_per_month": "mean",
        "outgoing_sms_per_month": "mean",
        "monthly_mb": "mean",
        "use_id": "count"
    }))
#Get Memoery usage
result.info(memory_usage='deep')
```

Describe

Descriptive statistics (mean, standard deviation, number of observations, minimum, maximum, and quartiles) of numerical columns can be calculated using the .describe() method, which returns a pandas dataframe of descriptive statistics.

```
import pandas as pd

data = pd.DataFrame({'A': [1, 2, 1, 4, 3, 5, 2, 3, 4, 1],
                     'B': [12, 14, 11, 16, 18, 18, 22, 13, 21, 17],
                     'C': ['a', 'a', 'b', 'a', 'b', 'c', 'b', 'a',
'b', 'a']})
print(data.describe())

#Note that since C is not a numerical column, it is excluded from
the output.
print(data['C'].describe())
```

Output:

```
                A          B
count  10.000000  10.000000
mean    2.600000  16.200000
std     1.429841   3.705851
min     1.000000  11.000000
25%     1.250000  13.250000
50%     2.500000  16.500000
75%     3.750000  18.000000
max     5.000000  22.000000
count     10
unique     3
top        a
freq       5
Name: C, dtype: object
```

EXAMPLE: DATA CLEANING USING PYTHON

Now, its time to apply our knowledge of Pandas on a project. We will use our knowledge gained so far to clean a dataset which we will use for further analysis. For this we have choosen hotel_booking dataset. The Hotel Booking demand dataset contains booking information for a city hotel and a resort hotel. It includes information such as booking time, length of stay, number of adults, children/babies, number of available parking spaces, among other things. Let's get started.

Import the data and briefly examine the dataset:

```
import pandas as pd
import numpy as np

#Import the dataset
```

```
df =
pd.read_csv("https://raw.githubusercontent.com/swapnilsaurav/Dataset/master/hotel_bookings.csv")

# brief look at the data
# shape and data types of the data
print(df.shape)
print(df.dtypes)

# select numeric columns
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
print(numeric_cols)

# select non numeric columns
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
print(non_numeric_cols)
```

From these results, we learn that the dataset has 119,390 rows and 32 columns. We also identify whether the features are numeric or categorical variables. These are all useful information which we will use later.

Now we will analyze and clean the dataset step by step.

1. Missing Data

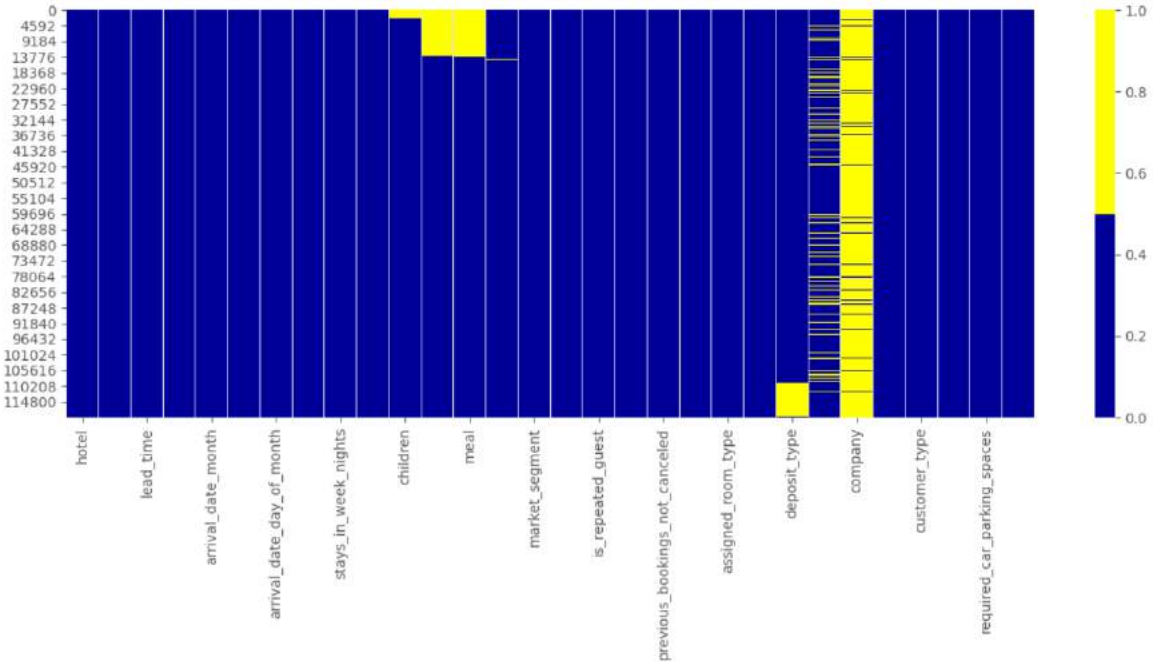
Dealing with missing data/value is one of the most tricky but common parts of data cleaning. While many models can live with other problems of the data, most models don't accept missing data.

We cover three techniques to learn more about missing data in our dataset.

Technique #1: Missing Data Heatmap

When there is a smaller number of features, we can visualize the missing data via heatmap.

```
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
plt.style.use('ggplot')
from matplotlib.pyplot import figure
cols = df.columns[:30] # first 30 columns
colours = ['#000099', '#ffff00'] # specify the colours - yellow is missing. blue is not missing.
sns.heatmap(df[cols].isnull(), cmap=sns.color_palette(colours))
plt.show()
```



The chart above demonstrates the missing data patterns of the first 30 features. The horizontal axis shows the feature name; the vertical axis shows the number of observations/rows; the yellow color represents the missing data while the blue color otherwise.
For example, we see that the company feature has missing values throughout many rows. While the *Children* feature only has little missing values around first 13000 rows.

Technique #2: Missing Data Percentage List
When there are many features in the dataset, we can make a list of missing data % for each feature. For if a larger dataset and the visualization can take too long to plot the graph, instead missing data % can be calculated using this code:

```
#Technique #2: Missing Data Percentage List
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())
    print('{} - {}%'.format(col, round(pct_missing*100)))
```

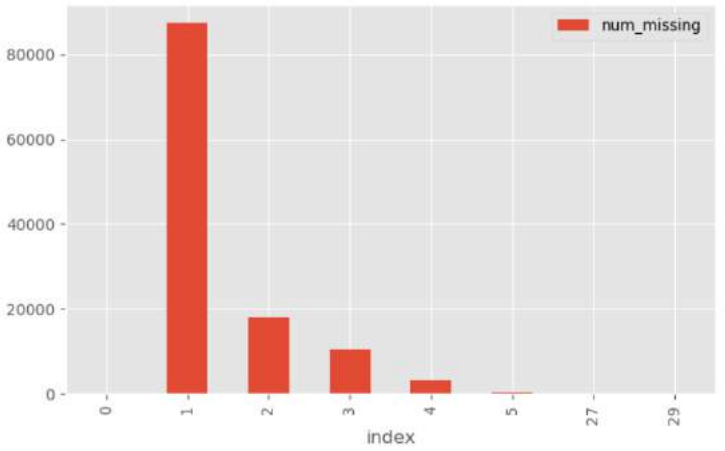
This produces a list below showing the percentage of missing values for each of the features. Specifically, we see that the company feature has 94% missing, while Children have only 2% missing. This list is a useful summary that can complement the heatmap visualization.

```
adults - 0%
children - 2%
babies - 11%
meal - 11%
country - 0%
market_segment - 0%
distribution_channel - 0%
is_repeated_guest - 0%
previous_cancellations - 0%
previous_bookings_not_canceled - 0%
reserved_room_type - 0%
assigned_room_type - 0%
booking_changes - 0%
deposit_type - 8%
agent - 14%
company - 94%
days_in_waiting_list - 0%
customer_type - 0%
```

Technique #3: Missing Data Histogram

Missing data histogram is also a technique for when we have many features. To learn more about the missing value patterns among observations, we can visualize it by a histogram.

```
#Technique #3: Missing Data Histogram
# first create missing indicator for features with missing data
for col in df.columns:
        missing = df[col].isnull()
        num_missing = np.sum(missing)

        if num_missing > 0:
            print('created missing indicator for: {}'.format(col))
            df['{}_ismissing'.format(col)] = missing

# then based on the indicator, plot the histogram of missing
values
ismissing_cols = [col for col in df.columns if 'ismissing' in col]
df['num_missing'] = df[ismissing_cols].sum(axis=1)

df['num_missing'].value_counts().reset_index().sort_values(by='ind
ex').plot.bar(x='index', y='num_missing')
plt.show()
```



This histogram helps to identify the missing values situations among the given observations.

Strategies to handle the missing values

There are no one solution to dealing with missing data. We have to study the specific feature and devise strategy to handle the data in best possible way for each feature. Below covers the four most common methods of handling missing data. But, if the situation is more complicated than usual, we need to be creative to use more sophisticated methods such as missing data modeling.

Solution #1: Drop the Observation

In statistics, this method is called the listwise deletion technique. In this solution, we drop the entire observation (row) as long as it contains a missing value. Only if we are sure that the missing data is not informative, we perform this. Otherwise, we should consider other solutions. There could be other criteria to use to drop the observations.

For example, from the missing data histogram, we notice that only a minimal amount of observations have over 12 features missing altogether. We may create a new dataset df_less_missing_rows deleting observations with over 12 missing features (out of total 32).

```
# drop rows with a lot of missing values.
ind_missing = df[df['num_missing'] > 12].index
print("Rows with more than 12 Missing values: \n",ind_missing)
df = df.drop(ind_missing, axis=0)
```

```
Rows with more than 12 Missing values:
 Int64Index([160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172,
             173],
            dtype='int64')
```

Solution #2: Drop the Feature

Similar to Solution #1, we only do this when we are confident that this feature doesn't provide useful information. For example, from the missing data % list, we notice that Company has a high missing value percentage of 94%. We may drop the entire feature.

```
# company has a lot of missing.
# If we want to drop.
cols_to_drop = ['company']
df = df.drop(cols_to_drop, axis=1)
```

Solution #3: Impute the Missing

We have dropped or removed the feature or the observation which couldn't have made much impact on our data. Now we are left with data which we cant remove as we might lose good info for our analysis. There are two popular options that people follow in these cases:

Option1: When the feature is a numeric variable, we can conduct missing data imputation. We replace the missing values with the average or median value from the data of the same feature that is not missing. Let's see this strategy in solution 3.

Option2: For categorical features, we can add a new category with a value such as "_MISSING_". For numerical features, we can replace it with a particular value such as -999. We will discuss this option as Solution 4 next.

Implementing option 1 - When the feature is a categorical variable, we may impute the missing data by the mode (the most frequent value).

```
##Checking: Missing Data Percentage List
print("Percentage of missing data after technique 1 and 2:")
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())
    if pct_missing > 0:
        print('{} - {:.3f}%'.format(col, round(pct_missing*100)))
```

Percentage of missing data after technique 1 and 2:

```
Percentage of missing data after technique 1 and 2:
children - 2.0%
babies - 11.0%
meal - 11.0%
country - 0.0%
deposit_type - 8.0%
agent - 14.0%
```

Looking at the data, lets plan a strategy to replace the missing values. Country might have very few missing values still that's why its appearing on our list. Let's see the data types of the columns:

```
# Get current data type of columns
print(df.dtypes)
#Convert meal, deposit_type and agent to categorical type
df['meal'] = pd.Categorical(df.meal)
df['deposit_type'] = pd.Categorical(df.deposit_type)
df['agent'] = pd.Categorical(df.agent)
```

Data Missing	Data Type	Imputation Strategy
children - 2.0%	Numeric	Replace with median of the same feature
babies - 11.0%	Numeric	Replace with median of the same feature
meal- 11.0%	Categorical	Replace with mode of the same feature
country - 0.0%	Categorical	Replace with mode of the same feature
deposit_type - 8.0%	Categorical	Replace with mode of the same feature
agent - 14.0%	Categorical	Replace with mode of the same feature

Replace each numeric feature's missing values with the median:

```
# replace missing values with the median.
med = df['children'].median()
df['children'] = df['children'].fillna(med)
med = df['babies'].median()
df['babies'] = df['babies'].fillna(med)
```

OR

When you have many features to be replaced, we can use a FOR loop as demonstrated below:

```
# impute the missing values and create the missing value indicator
# variables for each numeric column.
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
for col in numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)

    if num_missing > 0: #imputation for columns with missing value
        print('imputing missing values for: {}'.format(col))
        df['{}_ismissing'.format(col)] = missing
        med = df[col].median()
        df[col] = df[col].fillna(med)
```

Output:

```
imputing missing values for: children
imputing missing values for: babies
```

Now, lets turn our attention to replace the categorical missing values with the mode:

```
# impute the missing values and create the missing value
# indicator variables for each non-numeric column.
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values

for col in non_numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)

    if num_missing > 0:  # only do the imputation missing value
columns
        print('imputing missing values for: {}'.format(col))
        df['{}_ismissing'.format(col)] = missing

        top = df[col].describe()['top']  # impute with the most
frequent value.
        df[col] = df[col].fillna(top)
```

```
imputing missing values for: meal
imputing missing values for: country
imputing missing values for: deposit_type
imputing missing values for: agent
```

Solution 4: Replacing the missing values – option 2

For categorical features, we can add a new category with a value such as "_MISSING_". For numerical features, we can replace it with a particular value such as -999. This way, we are still keeping the missing values as valuable information.

```
# categorical
df['country'] = df['country'].fillna('_MISSING_')

# numeric
df['babies'] = df['babies'].fillna(-999)
```

To proceed further, we need to drop all the missing indicators we added to the data. We then plot heatmap again to see the missing data and save the data in a file to inspect. These steps are totally optional; I am doing only to see the result:

```
#Drop all the missing indicator columns
ismissing_cols = [col for col in df.columns if 'ismissing' in col]
df = df.drop(ismissing_cols, axis=1)
# plot the histogram of missing values
ismissing_cols = [col for col in df.columns if 'ismissing' in col]
df['num_missing'] = df[ismissing_cols].sum(axis=1)
df['num_missing'].value_counts().reset_index().sort_values(by='ind
ex').plot.bar(x='index', y='num_missing')
plt.show()
```

```
#Saving data to csv
df.to_csv('AfterMissingVal.csv', index=False)
```

2. Identifying Outiers (Irregular data)

What is an outlier data and why should we worry about it? Outliers are data that is distinctively different from other observations. They could be real outliers or mistakes. A value that "lies outside" (is much smaller or larger than) most of the other values in a set of data. For example in the scores 25,29,3,32,85,33,27,28 both 3 and 85 are "outliers".

Most likely outliers are mistakes which might have crept into the dataset. Sometimes, its easy to know if the outliers are mistakes and at times we may not know and we have to follow our instinct to handle it.

How to find out?
Depending on whether the feature is numeric or categorical, we can use different techniques to study its distribution to detect outliers.

Technique #1: Histogram/Box Plot
When the feature is numeric, we can use a histogram and box plot to detect outliers.
Below is the histogram of feature *total_of_special_requests*.

```
# histogram of total_of_special_requests.
df['total_of_special_requests'].hist(bins=100)
```

We can see if the distribution (Histogram) is Left or Right skewed. For details on the analysis, please refer the chapter 3 and the section - Statistics relevant for descriptive data analysis.

To study the feature closer, let's make a box plot.

```
df.boxplot(column=['total_of_special_requests'])
plt.show()
```

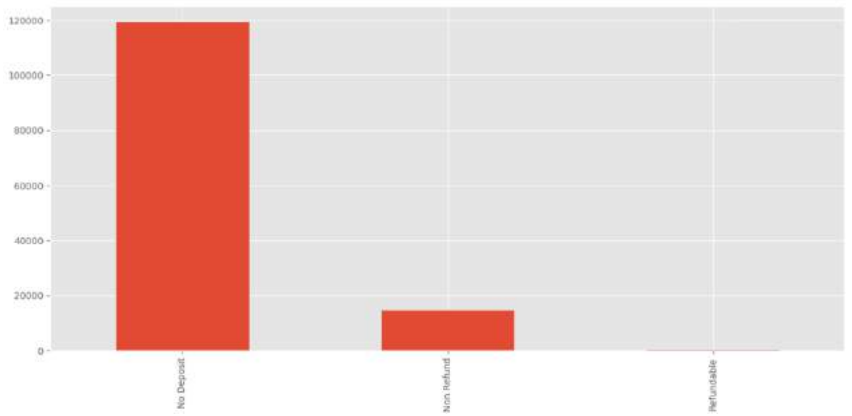
Boxplot shows outliers on the graph as dots.

Technique #2: Descriptive Statistics
For numeric features, the outliers could be too distinct that the box plot can't visualize them. Instead, we can look at their descriptive statistics.

```
print(df['total_of_special_requests'].describe())
```

Technique #3: Bar Chart
When the feature is categorical, we can use a bar chart to learn about its categories and distribution. For example, when we plot the bar graph for the values of "deposit_type", we get values like No Deposit, Refundable and Non Refund. If we see values like "Missing", "Not Known", "Not Available" would mean outliers.

```
# bar chart- distribution of a categorical variable
df['deposit_type'].value_counts().plot.bar(x='deposit type',
y='count')
plt.show()
```



Luckily in our case here, we don't have any outliers.
You can also use other techniques like scatter plot, z-score, and clustering.

While outliers are not hard to detect, we have to determine the right solutions to handle them. It highly depends on the dataset and the goal of the project. The methods of handling outliers are somewhat similar to missing data. We either drop or adjust or keep them. We can refer back to the missing data section for possible solutions.

3. Unnecessary data

After all the hard work done for missing data and outliers, let's look at unnecessary data, which is more straightforward. All the data feeding into the model should serve the purpose of the project. The unnecessary data is when the data doesn't add value. We cover three main types of unnecessary data due to different reasons.

Unnecessary type #1: Uninformative / Repetitive

Sometimes one feature is uninformative because it has too many rows being the same value. How to find out? We can create a list of features with a high percentage of the same value.
For example, we specify below to show features with over 95% rows being the same value.

```
num_rows = len(df.index)
low_information_cols = []  #

for col in df.columns:
    cnts = df[col].value_counts(dropna=False)
    top_pct = (cnts / num_rows).iloc[0]

    if top_pct > 0.95:
```

```
        low_information_cols.append(col)
        print('{0}: {1:.5f}%'.format(col, top_pct * 100))
        print(cnts)
        print(
```

Output: just a screenshot of 3 data put in 3 columns for better readibilty:

```
babies: 99.36336%
0.0     118616
1.0        746
2.0         12
9.0          1
10.0         1
Name: babies, dtype: int64
```

```
is_repeated_guest: 96.80840%
0.0    115566
1.0      3810
Name: is_repeated_guest, dtype: int64
```

```
previous_bookings_not_canceled: 96.96756%
0.0     115756
1.0       1542
2.0        580
3.0        333
4.0        229
```

We need to understand the reasons behind the repetitive feature. When they are genuinely uninformative, we can simply remove them from our dataframe.

Unnecessary type #2: Irrelevant

Here we are talking about the feature (column) relevant to our business objective we are trying to solve. If the features are not related to the question we are trying to solve in the project, they are irrelevant.

How to find out? We need to skim through the features to identify irrelevant ones. For example, a feature “reservation_status_date” may not be relevant to know How many daysthe customer is going to stay in the hotel. When the features are not serving the project’s goal, we can remove them.

Unnecessary type #3: Duplicates

The duplicate data is when copies of the same observation exist. There are two main types of duplicate data: All Features based and Key Features based.

Duplicates type #1: All Features based

This duplicate happens when all the features’ values within the observations are the same. It is easy to find. We first remove the unique identifier **id** in the dataset. Then we create a dataset called df_dupdropped by dropping the duplicates. We compare the shapes of the two datasets (df and df_ dupdropped) to find out the number of duplicated rows.

```
# we know that column 'id' is unique, but what if we drop it?
dupdropped = df.drop('id', axis=1).drop_duplicates()

# there were duplicate rows
print(df.shape)
print(dupdropped.shape)
```

```
(119376, 33)
(87372, 32)
```

One column less is the ID column which has been removed but look at the number of rows – it has fallen from 119,376 to 87,372 – we have removed 32,004 rows.
What do we do? Again take a call based on your business case. Maybe for a machine learning algorithm and predictive analytics duplicates may not be relevant but for data visualization, you might use it for your purpose. Take your call.

Duplicates type #2: Key Features based
Sometimes it is better to remove duplicate data based on a set of unique identifiers. For example, the chances of two transactions happening at the same time, with the same square footage, the same price, and the same build year are close to zero.

In our dataset, we can set up a group of critical features as unique identifiers for transactions. We include reservation_status_date, is_repeated_guest, previous_cancellations, previous_bookings_not_canceled, days_in_waiting_list. We check if there are duplicates based on them. The result will have observations more than or equal compared to technique 1 of all features.

```
# check if there are duplicates based on given key features
key = ['reservation_status_date', 'distribution_channel',
'market_segment', 'reserved_room_type', 'assigned_room_type',
'booking_changes', 'deposit_type']
df.fillna(999).groupby(key)['id'].count().sort_values(ascending=False).head(20)

# drop duplicates based on an subset of variables.
key = ['reservation_status_date','is_repeated_guest',
'previous_cancellations', 'previous_bookings_not_canceled',
'days_in_waiting_list']
df_dedupped2 = df.drop_duplicates(subset=key)

print(df.shape)
print(df_dedupped2.shape)
```

4. Inconsistent data

We need to explore the data in different ways to find out the inconsistent data. Much of the time, it depends on observations and experience. There is no set code to run and fix them all.
Here we cover four inconsistent data types.

Inconsistent type #1: Capitalization
Inconsistent usage of upper and lower cases in categorical values is a common mistake. It could cause issues since analysis in Python is case sensitive. But sometimes there is inconsistent capitalization usage within the same feature. The “check-out”, “Check-out” and “Check-Out” could refer to the same value. To avoid this, we can put all letters to lower cases (or upper cases or Title cases).

```
#Lets look at reservation_status feature
print(df['reservation_status'].value_counts(dropna=False))
# make everything lower case.
df['reservation_status_lower'] =
df['reservation_status'].str.lower()
print(df['reservation_status_lower'].value_counts(dropna=False))
```

Output:

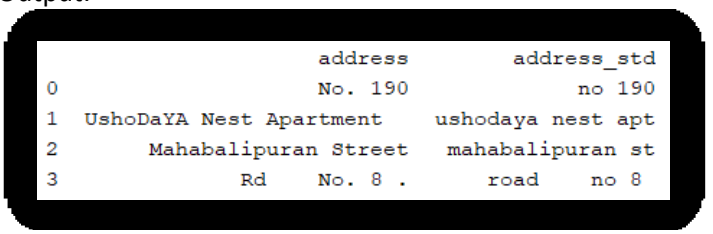
```
Check-Out    75153
Canceled     43016
No-Show       1207
Name: reservation_status, dtype: int64
check-out    75153
canceled     43016
no-show       1207
Name: reservation_status_lower, dtype: int64
```

Inconsistent type #2: Addresses

The address feature could be a headache for many of us. Because people entering the data into the database often don't follow a standard format. We can find messy address data by looking at it. Even though sometimes we can't spot any issues, we can still run code to standardize them. There is no address column in our dataset for privacy reasons. So we create a new dataset df_add_ex with feature address. How do we handle it? We run the below code to lowercase the letters, remove white space, delete periods and standardize wordings.

```
# no address column in the hotel_booking dataset.
# So create one to show the code.
df_add_ex = pd.DataFrame(['No. 190', 'UshoDaYA Nest Apartment  ',
'Mahabalipuran Street', '  Rd    No. 8 .'], columns=['address'])
print(df_add_ex)
#Standardise the address
df_add_ex['address_std'] = df_add_ex['address'].str.lower()
df_add_ex['address_std'] = df_add_ex['address_std'].str.strip() #
remove leading and trailing whitespace.
df_add_ex['address_std'] =
df_add_ex['address_std'].str.replace('\\.', '') # remove period.
df_add_ex['address_std'] =
df_add_ex['address_std'].str.replace('\\bstreet\\b', 'st') #
replace street with st.
df_add_ex['address_std'] =
df_add_ex['address_std'].str.replace('\\bapartment\\b', 'apt') #
replace apartment with apt.
df_add_ex['address_std'] =
df_add_ex['address_std'].str.replace('\\brd\\b', 'road') # replace
apartment with apt.
print(df_add_ex)
```

Output:



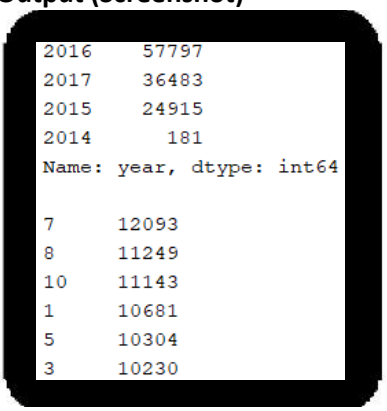
	address	address_std
0	No. 190	no 190
1	UshoDaYA Nest Apartment	ushodaya nest apt
2	Mahabalipuran Street	mahabalipuran st
3	Rd No. 8 .	road no 8

Inconsistent type #3: Feature Formats

Another standardization we need to perform is the data formats. One example is to convert the feature from string to DateTime format. The feature reservation_status_date is in string format while it represents dates. We can convert it and extract the date or time values by using the code below. After this, it's easier to analyze the transaction volume group by either year or month.

```
df['reservation_status_date_dt'] =
pd.to_datetime(df['reservation_status_date'], format='%d-%m-%Y')
df['year'] = df['reservation_status_date_dt'].dt.year
df['month'] = df['reservation_status_date_dt'].dt.month
df['weekday'] = df['reservation_status_date_dt'].dt.weekday

print(df['year'].value_counts(dropna=False))
print()
print(df['month'].value_counts(dropna=False))
```

Output (Screenshot)



```
2016    57797
2017    36483
2015    24915
2014      181
Name: year, dtype: int64

7     12093
8     11249
10    11143
1     10681
5     10304
3     10230
```

Inconsistent type #4: Categorical Values

A categorical feature has a limited number of values. Sometimes there may be other values due to reasons such as typos. Let's create a new dataset to this since our dataset do not have such problem. For instance, the value of city was typed by mistakes as "delhiii" and "dalhi". But they both refer to the correct value "delhi". A simple way to identify them is fuzzy logic (or edit

distance). It measures how many letters (distance) we need to change the spelling of one value to match with another value.

We know that the categories should only have four values of “delhi”, “mumbai”, “hyderabad”, and “kolkata”. We calculate the distance between all the values and the word “delhi” (and “mumbai”). We can see that the ones likely to be typos have a smaller distance with the correct word. Since they only differ by few of letters. For this example, we will use nltk package, a popular package to perform Natural Language Processing.

```
from nltk.metrics import edit_distance

df_city_ex = pd.DataFrame(data={'city': ['delhiii', 'dalhi',
'delhi', 'mumbai', 'moombai', 'mumbhai', 'hyderabad', 'kolkata']})
df_city_ex['city_distance_delhi'] = df_city_ex['city'].map(lambda
x: edit_distance(x, 'delhi'))
df_city_ex['city_distance_mumbai'] = df_city_ex['city'].map(lambda
x: edit_distance(x, 'mumbai'))
print(df_city_ex)
```

Output:

	city	city_distance_delhi	city_distance_mumbai
0	delhiii	2	6
1	dalhi	1	5
2	delhi	0	5
3	mumbai	5	0
4	moombai	6	2
5	mumbhai	5	1
6	hyderabad	7	7
7	kolkata	6	6

Resolution: We can set criteria to convert these typos to the correct values. For example, the below code sets all the values within 2 letters distance from “delhi” to be “delhi”. Similarly, we have for mumbai to mumbai.

```
msk = df_city_ex['city_distance_delhi'] <= 2
df_city_ex.loc[msk, 'city'] = 'delhi'
msk = df_city_ex['city_distance_mumbai'] <= 2
df_city_ex.loc[msk, 'city'] = 'mumbai'

df_city_ex['city_distance_delhi'] = df_city_ex['city'].map(lambda
x: edit_distance(x, 'delhi'))
df_city_ex['city_distance_mumbai'] = df_city_ex['city'].map(lambda
x: edit_distance(x, 'mumbai'))
print(df_city_ex)
```

Output:

	city	city_distance_delhi	city_distance_mumbai
0	delhi	0	5
1	delhi	0	5
2	delhi	0	5
3	mumbai	5	0
4	mumbai	5	0
5	mumbai	5	0
6	hyderabad	7	7
7	kolkata	6	6

Hopefully, these techniques will help to clear all the “dirty” data that’s blocking your way to fit the data to analyze (and also to model your machine learning algorithm). To summarize, in this chapter we practiced following data cleaning processes:

- Removing unnecessary variables
- Deleting duplicate rows/observations
- Addressing outliers or invalid data
- Dealing with missing values
- Standardizing or categorizing values
- Correcting typographical errors

Data cleaning tasks will be very dependent on the dataset that you’re working with. You may not be performing all of the tasks mentioned here. What do we do after cleaning the data? That’s what we will learn in the next chapter.

UNIT 5: PREPARE YOUR DATA

This chapter should be seen as a continous of chapter 4. What we did (data cleaning) is part of data preparation. There are some topics which will be repeated here too but understand that these two chapters should be seen as part of same topic.

GETTING STARTED: DATA PREPROCESSING USING SKLEARN

Data preprocessing is an umbrella term that covers an array of operations data scientists will use to get their data into a form more appropriate for what they want to do with it. For example, before performing sentiment analysis of twitter data, you may want to strip out any html tags, white spaces, expand abbreviations and split the tweets into lists of the words they contain. When analyzing spatial data, you may scale it so that it is unit-independent, that is, so that your algorithm doesn't care whether the original measurements were in miles or centimeters. Preprocessing is important step and we have to do it before we start machine learning to make sure that there is no error that gets into the model because of the data we have. This may be the most boring part but very crucial so that we get our analysis right! I have purposely put the headers as a separate section because we will have to repeat them every time we work on performing analysis. Let's get started.

GET THE DATASET

Step 1 is to get the dataset to work on the analysis. The dataset to be worked with this book has been placed on the Github location:
https://github.com/swapnilsaurav/MachineLearning

We will mention the filename of the dataset that can be downloaded from the above location for each of the exercise as we go along. For PreProcessing exercise, download the file - 1_Data_PreProcessing.csv from the above location. This dataset contains four columns – Regions (Region), number of Salesperson (Salesperson) in that region, Quotation that was given for a contract (Quotation) and if the team was awarded the contract or not (Win). The data from multiple contract is presented together hence you will see the regions are repeated. We have 14 observations. Before we proceed with the analysis, we will have to differentiate between dependent variables and the independent variables. In this example, the independent variables are the first 3 columns – Region, Salesperson and Quotation and dependent variable is Win. In our learning of the machine learning, we will use the independent variable to predict the dependent variable. We will use Region, Salesperson and Quotation column to predict if the contract can be won or not.

Region	Salesperson	Quotation	Win
North	44	72000	No
South	27	48000	Yes
East	30	54000	No

Figure: Dataset snapshot

IMPORT THE LIBRARY

The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the data analysis.

STANDARDIZATION, OR MEAN REMOVAL AND VARIANCE SCALING (FEATURE SCALING)

Looking at the dataset, the Salesperson independent variable value varies from 27 to 48 and the Quotation value varies from 40000 to 90000. In scenarios like these, owing to mere greater numeric range, the impact on response variables by the feature having greater numeric range could be more than the one having less numeric range, and this could, in turn, impact prediction accuracy. The objective is to improve predictive accuracy and not allow a particular feature impact the prediction due to large numeric value range. Thus, we may need to normalize or scale values before proceeding under different features such that they fall under common range.

There are couple of ways to scale the value:

- Min-Max Normalization: Data frame could be normalized using Min-Max normalization technique which specify following formula to be applied on each value of features to be normalized: (X - min(X))/(max(X) - min(X))
- Z-Score Normalization: The disadvantage with min-max normalization technique is that it tends to bring data towards the mean. If there is a need for outliers to get weighted more than the other values, z-score standardization technique suits better.
  Formula is: (X - mean(X))/(Std. Dev.(X))

The function scale provides a quick and easy way to perform this operation on a single array-like dataset but before install the sklearn package using: **Python37-32\Scripts>pip install sklearn** on command prompt.

```
from sklearn import preprocessing
#import sklearn as sk
import numpy as np
X_train = np.array([[ 0, -35,  25],
                    [ 200,  0,  20],
                    [ 30,  10, 0]])
X_scaled = preprocessing.scale(X_train)
print(X_scaled)

#Calculate Mean of the Scaled values
print(X_scaled.mean(axis=0))
#Calculate Standard Deviation of the Scaled values
print(X_scaled.std(axis=0))
```

Output:

[[-0.87056284 -1.38218948 0.9258201]

```
[ 1.40047065  0.43193421  0.46291005]
 [-0.52990781  0.95025527 -1.38873015]
```

Scaled data has zero mean and unit variance.

Scaling Feature to a range

An alternative standardization is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. This can be achieved using MinMaxScaler or MaxAbsScaler, respectively.

Here is an example to scale a data matrix to the [0, 1] range:

```
from sklearn import preprocessing
#import sklearn as sk
import numpy as np
X_train = np.array([[ 0, -35,  25],
                    [ 200,  0,  20],
                    [ 30,  10, 0]])
min max scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
print(X_train_minmax)
```

Output:

```
[[0.      0.      1.     ]
 [1.      0.77777778 0.8    ]
 [0.15    1.      0.     ]]
```

If MinMaxScaler is given an explicit feature_range=(min, max) the full formula is:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

MaxAbsScaler works in a very similar fashion, but scales in a way that the training data lies within the range [-1, 1] by dividing through the largest maximum value in each feature. It is meant for data that is already centered at zero or sparse data.

```
from sklearn import preprocessing
#import sklearn as sk
import numpy as np
X_train = np.array([[ 0, -35,  25],
                    [ 200,  0,  20],
                    [ 30,  10, 0]])
max_abs_scaler = preprocessing.MaxAbsScaler()
X_train_maxabs = max_abs_scaler.fit_transform(X_train)
print(X_train_maxabs)
```

Example using StandardScaler:

```
from sklearn.preprocessing import StandardScaler
data = [[0, 0], [0, 0], [1, 1], [1, 1]]
scaler = StandardScaler()
print(scaler.fit(data))
# StandardScaler(copy=True, with_mean=True, with_std=True)
#Get the transformed data
print(scaler.transform(data))
print(scaler.mean_)
print(scaler.transform([[2, 2]]))
```

MISSING DATA AND SPARSE MATRICES

For various reasons, many real world datasets contain missing values, often encoded as blanks, NaNs or other placeholders. Such datasets however are incompatible with scikit-learn estimators which assume that all values in an array are numerical, and that all have and hold meaning. This happens very frequently while working with the real time dataset hence you need to learn the trick to handle missing data and make it look good so that machine learning algorithm can run correctly. As you can see in the current dataset, we have 2 missing data – one under Quotation and other under Salesperson. One option is to remove the missing data altogether from the analysis but that's not the right approach because this comes at the price of losing data which may be valuable (even though incomplete). Another option which is the most common option is to be replace the missing data with the mean of all the other value. Let's understand how sklearn package handles it.

A better strategy is to impute the missing values, i.e., to infer them from the known part of the data. The SimpleImputer class provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located. This class also allows for different missing values encodings.

The following snippet demonstrates how to replace missing values, encoded as np.nan, using the mean value of the columns (axis 0) that contain the missing values:

```
import numpy as np
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
print(imp.fit([[1, 2], [np.nan, 3], [7, 6]]))
#SimpleImputer(copy=True, fill_value=None, missing_values=nan,
strategy='mean', verbose=0)
X = [[np.nan, 2], [6, np.nan], [7, 6]]
print(imp.transform(X))
```

Output:

```
SimpleImputer(copy=True, fill_value=None, missing_values=nan, strategy='mean',
    verbose=0)
[[4.       2.       ]
 [6.       3.66666667]
 [7.       6.       ]]
```

Note: 4 is because of mean(1,NaN,7) and mean of (2,3,6) of fit model

Sparse matrices are dataset with zero or not usable values like negatives. The SimpleImputer class also supports sparse matrices:

```
import scipy.sparse as sp
from sklearn.impute import SimpleImputer
X = sp.csc_matrix([[1, 2], [0, -1], [8, 4]])
imp = SimpleImputer(missing_values=-1, strategy='mean')
imp.fit(X)
#SimpleImputer(copy=True, fill_value=None, missing_values=-1,
strategy='mean', verbose=0)
X_test = sp.csc_matrix([[-1, 2], [6, -1], [7, 6]])
print(imp.transform(X_test).toarray())
```

Output:

```
[[3. 2.]
 [6. 3.]
 [7. 6.]]
```

HANDLING OUTLIERS

In Data Science, an Outlier is an observation point that is distant from other observations. An Outlier may be due to variability in the measurement or it may indicate experimental error. Outliers, being the most extreme observations, may include the sample maximum or sample minimum, or both, depending on whether they are extremely high or low. However, the sample maximum and minimum are not always outliers because they may not be unusually far from other observations. Let's look at an example of below:

Example 1: Let's look at the total score of Opening Batsmen at a Cricket World cup:

Player	Total Score
Player 1	450
Player 2	350
Player 3	400
Player 4	500
Player 5	475
Player 6	25
Player 7	1900
Player 8	350
Player 9	425
Player 10	550

As you can see from the above collected data that most of the players have scored between 350 and 550. There is also one score of 25 which looks very low and other 1900 which looks very high. These figures can be just a typing mistake or it is showing the variance in your data and indicating that Player6 is performing very bad and Player 7 is performing very well.
Now that we know outliers can either be a mistake or just variance, how would you decide if they are important or not. Well, it is pretty simple if they are the result of a mistake, then we can ignore them, but if it is just a variance in the data we would need think a bit further.
The outliers can be a result of a mistake during data collection or it can be just an indication of variance in your data. In either case, Outliers in data can distort predictions and affect the accuracy, if you don't detect and handle them. Before we try to understand whether to ignore the outliers or not, we need to know the ways to identify them.

For this section, we will be using Boston House Pricing Dataset which is included in the sklearn dataset API. We will load the dataset and separate out the features and targets.

Reading the dataset:

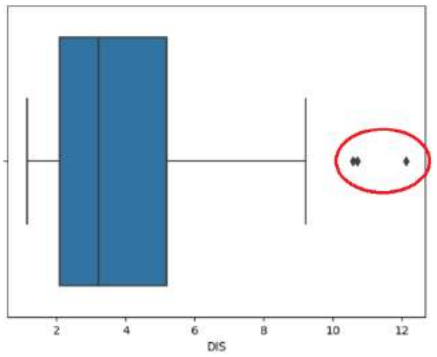
```
from sklearn.datasets import load_boston
import pandas as pd

boston = load_boston()
x = boston.data
y = boston.target
columns = boston.feature_names
#create the dataframe
boston_df = pd.DataFrame(boston.data)
boston_df.columns = columns
print(boston_df.head())
```

Discover outliers with visualization tools

Box plot

We have seen BoxPlot in Data Visualization chapter. In Box Plot, outliers may be plotted as individual points. Let's try with above dataset and see:

```
import seaborn as sns
from matplotlib import pyplot as plt
sns.boxplot(x=boston_df['DIS'])
plt.show()
```

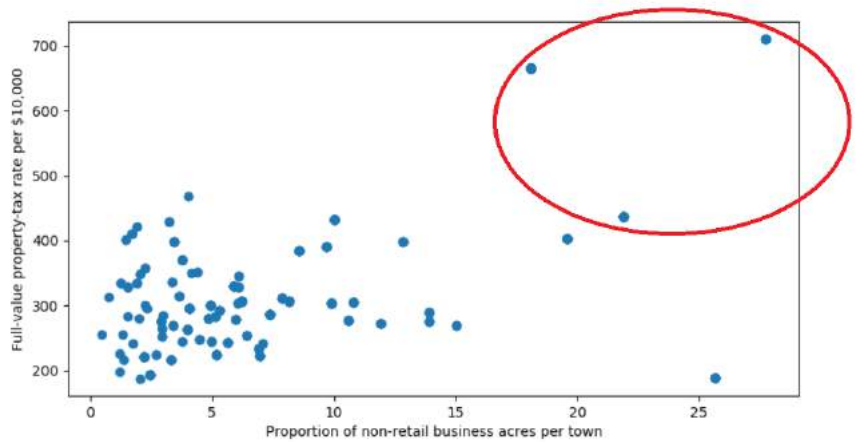


Above plot shows three points between 10 to 12, these are outliers as there are not included in the box of other observation i.e no where near the quartiles. Here we analysed Uni-variate outlier i.e. we used DIS column only to check the outlier. But we can do multivariate outlier analysis too. We can do the multivariate analysis with Box plot, if we have a categorical values that can be used with any continuous variable and do multivariate outlier analysis. As we do not have categorical value in our Boston Housing dataset, we can not do multivariate outlier analysis using box plot.

Scatter Plot

The scatter plot is the collection of points that shows values for two variables. We can try and draw scatter plot for two variables from our housing dataset.

```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(boston_df['INDUS'], boston_df['TAX'])
ax.set_xlabel('Proportion of non-retail business acres per
town')
ax.set_ylabel('Full-value property-tax rate per $10,000')
plt.show()
```



We can see that most of data points are lying bottom left side but there are points which are far from the population like top right corner which can be considered as outliers.

Discover outliers with mathematical function

Z-Score

The Z-score is the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured. While calculating the Z-score we re-scale and center the data and look for data points which are too far from zero. These data points which are way too far from zero will be treated as the outliers. In most of the cases a threshold of 3 or -3 is used i.e if the Z-score value is greater than or less than 3 or -3 respectively, that data point will be identified as outliers.

We will use Z-score function defined in scipy library to detect the outliers.

```
from scipy import stats
import numpy as np
z = np.abs(stats.zscore(boston_df))
print(z)
# From above output, it is difficult to say which data point is
an outlier.
# Let's define a threshold to identify an outlier.
threshold = 3
print(np.where(z > 3))
```

Second print function will give data points where z-score is greater than 3. In the output, the first array contains the list of row numbers and second array respective column numbers, which mean z[55][1] have a Z-score higher than 3.

You can test by:

```
print(z[55][1])
Output: 3.375038763517309
```

IQR score

Box plot use the interquartile range (IQR) method to display data and outliers(shape of the data) but in order to be get a list of identified outlier, we will need to use the mathematical formula and retrieve the outlier data. This is also called the midspread or middle 50%, or technically H-spread, is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles, IQR = Q3 − Q1.

It is a measure of the dispersion similar to standard deviation or variance, but is much more robust against outliers. IQR is somewhat similar to Z-score in terms of finding the distribution of data and then keeping some threshold to identify the outlier.

Let's find out we can box plot uses IQR and how we can use it to find the list of outliers as we did using Z-score calculation. First we will calculate IQR,

```
Q1 = boston_df.quantile(0.25)
Q3 = boston_df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

Here we will get IQR for each column. The below code will give an output with some true and false values. The data point where we have False that means these values are valid whereas True indicates presence of an outlier.

```
print(boston_df < (Q1 - 1.5 * IQR)) or (boston_df > (Q3 + 1.5 *
IQR))
```

Working with Outliers: Correcting, Removing

During data analysis when you detect the outlier one of most difficult decision could be how one should deal with the outlier. Should they remove them or correct them? Before we talk about this, we will have a look at few methods of removing the outliers.

Z-Score

In the previous section, we saw how one can detect the outlier using Z-score but now we want to remove or filter the outliers and get the clean data. This can be done with just one line code as we have already calculated the Z-score.

```
boston_df_o = boston_df[(z < 3).all(axis=1)]
```

So, above code removed around 90+ rows from the dataset i.e. outliers have been removed.

IQR Score

Just like Z-score we can use previously calculated IQR score to filter out the outliers by keeping only valid values.

```
boston_df_out = boston_df[~((boston_df < (Q1 - 1.5 * IQR))
|(boston_df > (Q3 + 1.5 * IQR))).any(axis=1)]
boston_df_out.shape
```

The above code will remove the outliers from the dataset. There are multiple ways to detect and remove the outliers but the methods, we have used for this exercise, are widely used. Whether an outlier should be removed or not. Every data analyst/data scientist might get these thoughts once in every problem they are working on and its best left to them to take a call. There can be multiple reasons you want to understand and correct the outliers.

CATEGORICAL DATA

Categorical variables represent types of data which may be divided into groups. In the current dataset we have 2 such variables – Region (categories are: North, South, East, West) and Win (categories are: Yes, No). Its important to convey to the machine learning algorithms about categorical values so that it doesn't treat them as regular values. The SimpleImputer class also supports categorical data represented as string values or pandas categoricals when using the 'most_frequent' or 'constant' strategy:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
df = pd.DataFrame([["a", "x"],
```

```
                        [np.nan, "y"],
                        ["a", np.nan],
                        ["b", "y"]], dtype="category")

imp = SimpleImputer(strategy="most_frequent")
print(imp.fit_transform(df))
```

Marking imputed values

The MissingIndicator transformer is useful to transform a dataset into corresponding binary matrix indicating the presence of missing values in the dataset. This transformation is useful in conjunction with imputation. When using imputation, preserving the information about which values had been missing can be informative. NaN is usually used as the placeholder for missing values. However, it enforces the data type to be float. The parameter missing_values allows to specify other placeholder such as integer. In the following example, we will use -1 as missing values:

```
import numpy as np
from sklearn.impute import MissingIndicator
X = np.array([[-1, -1, 1, 3],
                [4, -1, 0, -1],
                [8, -1, 1, 0]])
indicator = MissingIndicator(missing_values=-1)
mask_missing_values_only = indicator.fit_transform(X)
print(mask_missing_values_only)
```

Output:

```
[[ True  True False]
 [False  True  True]
 [False  True False]]
```

Now we create a FeatureUnion. All features will be imputed using SimpleImputer, in order to enable classifiers to work with this data. Additionally, it adds the indicator variables from MissingIndicator.

```
from sklearn.impute import SimpleImputer, MissingIndicator
from sklearn.pipeline import FeatureUnion

transformer = FeatureUnion(
     transformer_list=[
          ('features', SimpleImputer(strategy='mean')),
          ('indicators', MissingIndicator())])
transformer = transformer.fit(X_train, y_train)
results = transformer.transform(X_test)
print(results)
print(results.shape)
```

We will learn about X_train and y_train in the next section where we talk about training set and test set.

TRAINING SET AND TEST SET

We need to split our dataset into training set and test set for machine learning algorithm. We need to split the dataset because we want the machine to learn the algorithm and make prediction. We are going to build our machine learning algorithm on the training dataset and test it on the test set to know how well it has "learnt" the correlation.

Next question that is asked is how much of the data should be Training set and Test set. Best practice is to choose 80% of the data as Training set and 20% as Test set. Below is the example of splitting in train set and test set. In the examples running the models, we will see the application and understand better about it.

```
# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state = 0)
```

EXAMPLE USING 1_DATA_PREPROCESSING.CSV FILE

That was the last step in Data Preprocessing and now our data is ready to be used by Machine Learning algorithm. We have learnt all the basic steps in data preprocessing. We are not going to use all these steps always. It depends on the given dataset. Lets use all the pre-processing steps on our dataset.

```
# Data Preprocessing
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values

# Taking care of missing data
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values = 'NaN', strategy = 'mean',
axis = 0)
imputer = imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])


# Encoding categorical data
# Encoding the Independent Variable
from sklearn.preprocessing import LabelEncoder,
OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
```

```
onehotencoder = OneHotEncoder(categorical_features = [0])
X = onehotencoder.fit_transform(X).toarray()
# Encoding the Dependent Variable
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state = 0)

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""
```

OUTLIER DETECTION EXAMPLE USING SKLEARN.DATASETS.LOAD_BOSTON

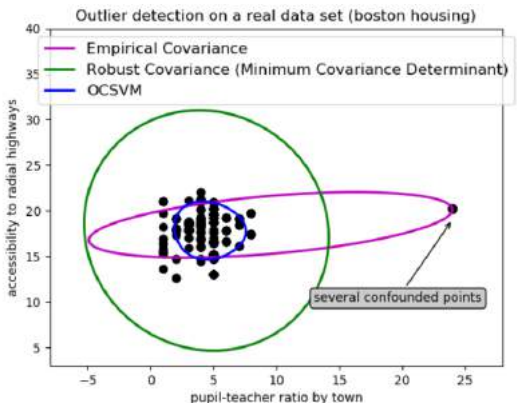
This example illustrates the need for robust covariance estimation on a real data set. It is useful both for outlier detection and for a better understanding of the data structure. We selected two sets of two variables from the Boston housing data set as an illustration of what kind of analysis can be done with several outlier detection tools. For the purpose of visualization, we are working with two-dimensional examples, but one should be aware that things are not so trivial in high-dimension, as it will be pointed out.

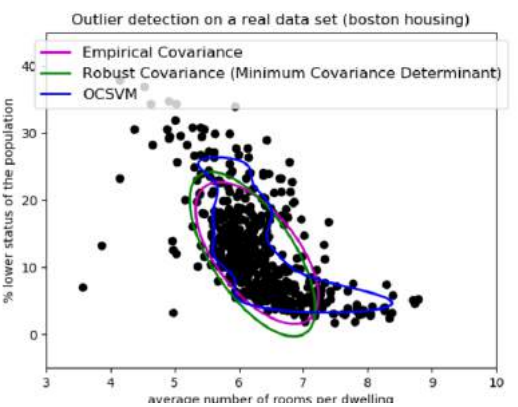
In both examples below, the main result is that the empirical covariance estimate, as a non-robust one, is highly influenced by the heterogeneous structure of the observations. Although the robust covariance estimate is able to focus on the main mode of the data distribution, it sticks to the assumption that the data should be Gaussian distributed, yielding some biased estimation of the data structure, but yet accurate to some extent. The One-Class SVM does not assume any parametric form of the data distribution and can therefore model the complex shape of the data much better.

```
import numpy as np
from sklearn.covariance import EllipticEnvelope
from sklearn.svm import OneClassSVM
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn.datasets import load_boston

# Get data
X1 = load_boston()['data'][:, [8, 10]]  # two clusters
X2 = load_boston()['data'][:, [5, 12]]  # "banana"-shaped

# Define "classifiers" to be used
```

```
classifiers = {
    "Empirical Covariance": EllipticEnvelope(support_fraction=1.,
                                             contamination=0.261),
    "Robust Covariance (Minimum Covariance Determinant)":
    EllipticEnvelope(contamination=0.261),
    "OCSVM": OneClassSVM(nu=0.261, gamma=0.05)}
colors = ['m', 'g', 'b']
legend1 = {}
legend2 = {}

# Learn a frontier for outlier detection with several classifiers
xx1, yy1 = np.meshgrid(np.linspace(-8, 28, 500), np.linspace(3, 40,
500))
xx2, yy2 = np.meshgrid(np.linspace(3, 10, 500), np.linspace(-5, 45,
500))
for i, (clf_name, clf) in enumerate(classifiers.items()):
    plt.figure(1)
    clf.fit(X1)
    Z1 = clf.decision_function(np.c_[xx1.ravel(), yy1.ravel()])
    Z1 = Z1.reshape(xx1.shape)
    legend1[clf_name] = plt.contour(
        xx1, yy1, Z1, levels=[0], linewidths=2, colors=colors[i])
    plt.figure(2)
    clf.fit(X2)
    Z2 = clf.decision_function(np.c_[xx2.ravel(), yy2.ravel()])
    Z2 = Z2.reshape(xx2.shape)
    legend2[clf_name] = plt.contour(
        xx2, yy2, Z2, levels=[0], linewidths=2, colors=colors[i])

legend1_values_list = list(legend1.values())
legend1_keys_list = list(legend1.keys())

# Plot the results (= shape of the data points cloud)
plt.figure(1)  # two clusters
plt.title("Outlier detection on a real data set (boston housing)")
plt.scatter(X1[:, 0], X1[:, 1], color='black')
bbox_args = dict(boxstyle="round", fc="0.8")
arrow_args = dict(arrowstyle="->")
plt.annotate("several confounded points", xy=(24, 19),
             xycoords="data", textcoords="data",
             xytext=(13, 10), bbox=bbox_args, arrowprops=arrow_args)
plt.xlim((xx1.min(), xx1.max()))
plt.ylim((yy1.min(), yy1.max()))
plt.legend((legend1_values_list[0].collections[0],
            legend1_values_list[1].collections[0],
            legend1_values_list[2].collections[0]),
           (legend1_keys_list[0], legend1_keys_list[1],
legend1_keys_list[2]),
           loc="upper center",
           prop=matplotlib.font_manager.FontProperties(size=12))
plt.ylabel("accessibility to radial highways")
plt.xlabel("pupil-teacher ratio by town")
```

```
legend2_values_list = list(legend2.values())
legend2_keys_list = list(legend2.keys())

plt.figure(2)  # "banana" shape
plt.title("Outlier detection on a real data set (boston housing)")
plt.scatter(X2[:, 0], X2[:, 1], color='black')
plt.xlim((xx2.min(), xx2.max()))
plt.ylim((yy2.min(), yy2.max()))
plt.legend((legend2_values_list[0].collections[0],
            legend2_values_list[1].collections[0],
            legend2_values_list[2].collections[0]),
           (legend2_keys_list[0], legend2_keys_list[1],
legend2_keys_list[2]),
           loc="upper center",
           prop=matplotlib.font_manager.FontProperties(size=12))
plt.ylabel("% lower status of the population")
plt.xlabel("average number of rooms per dwelling")

plt.show()
```





Covariance is a measure of the joint variability of two random variables. If the greater values of one variable mainly correspond with the greater values of the other variable, and the same holds for the lesser values, (i.e., the variables tend to show similar behavior), the covariance is positive. In the opposite case, when the greater values of one variable mainly correspond to the lesser values of the other, (i.e., the variables tend to show opposite behavior), the covariance is negative. The sign of the covariance therefore shows the tendency in the linear relationship between the variables. The magnitude of the covariance is not easy to interpret because it is not normalized and hence depends on the magnitudes of the variables. The normalized version of the covariance, the correlation coefficient, however, shows by its magnitude the strength of the linear relation.

In this example, we are using a one-class SVM for novelty detection. One-class SVM is an unsupervised algorithm that learns a decision function for novelty detection: classifying new data as similar or different to the training set.