UNIT-3

SYLLABUS

Cleansing, integrating, and transforming data: Cleansing data, correct errors as early as possible, combining data from different data sources, Transforming data

Exploratory data analysis, **Build the models**: Model and variable selection, Model execution, Model diagnostics and model comparison

Step 3: Cleansing, integrating, and transforming data: -

The data received from the data retrieval phase is likely to be **"a diamond in the rough."** Your task now is to sanitize and prepare it for use in the modeling and reporting phase. Doing so is tremendously important because your models will perform better and you'll lose less time trying to fix strange output. It can't be mentioned nearly enough times: **garbage in equals garbage out.** Your model needs the data in a specific format, so data transformation will always come into play. It's a good habit to correct data errors as early on in the process as possible. However, this isn't always possible in a realistic setting, so you'll need to take corrective actions in your program.

Figure 2.4 shows the most common actions to take during the data cleansing, integration, and transformation phase

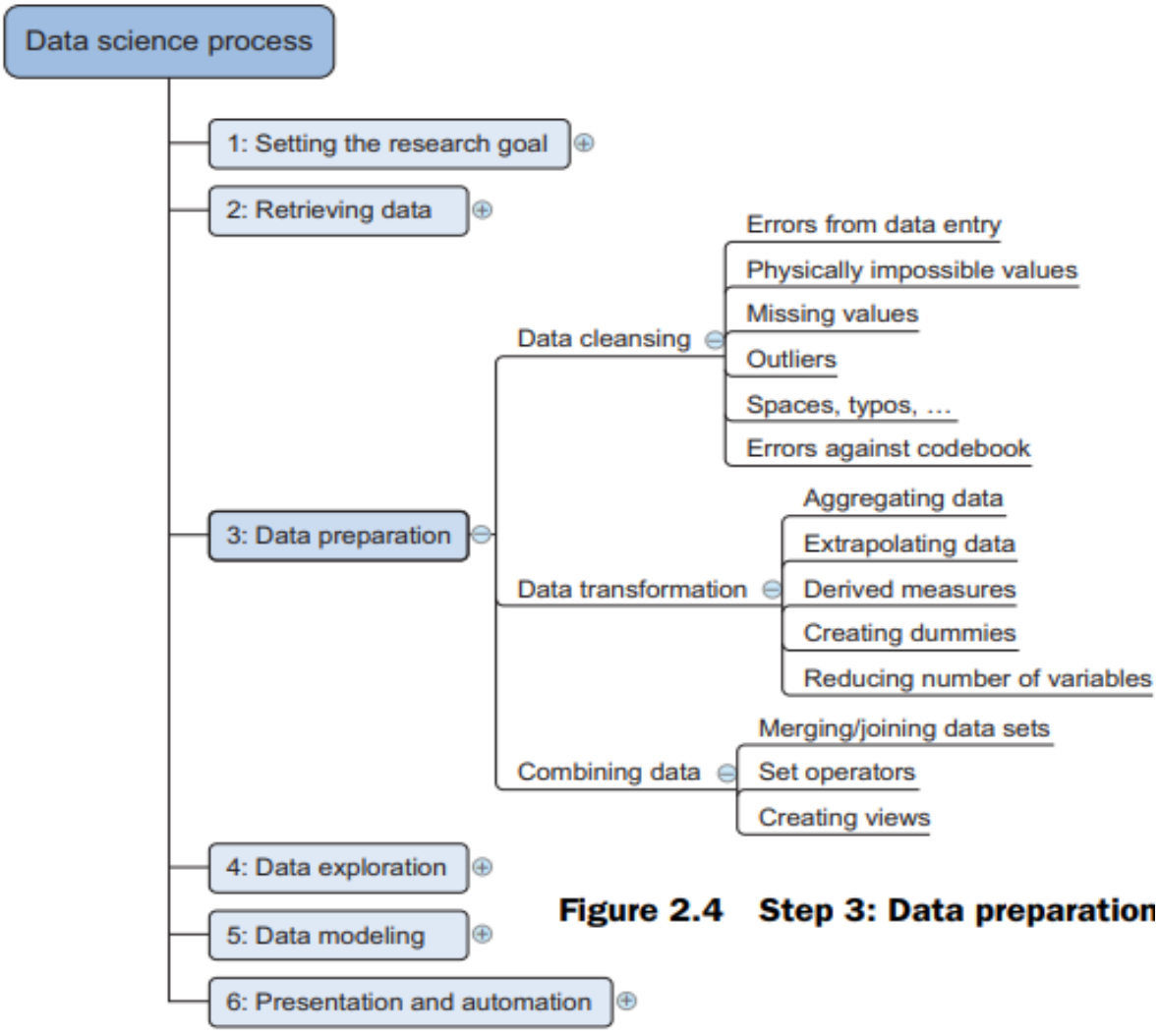


Figure 2.4 Step 3: Data preparation

Cleansing data: -

Data cleansing is a sub process of the data science process that focuses on removing errors in your data so your data becomes a true and consistent representation of the processes it originates from.

By "**true and consistent representation**" we imply that at least two types of errors exist. The first type is the **interpretation error**, such as when you take the value in your data for granted, like saying that a person's age is greater than 300 years. The second type of error points to **inconsistencies between data sources** or against your company's standardized values. An example of this class of errors is putting "Female" in one table and "F" in another when they represent the same thing: that the person is female. Another example is that you use Pounds in one table and Dollars in another. Too many possible errors exist for this list to be exhaustive, but table 2.2 shows an overview of the types of errors that can be detected with easy checks—the "low hanging fruit," as it were.

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

Table 2.2 An overview of common errors

Sometimes you'll use more advanced methods, such as simple modeling, to find and identify data errors; diagnostic plots can be especially insightful. For example, in figure 2.5 we use a measure to identify data points that seem out of place. We do a regression to get acquainted with the data and detect the influence of individual observations on the regression line. When a single observation has too much influence, this can point to an error in the data, but it can also be a valid point. At the data cleansing stage, these advanced methods are, however, rarely applied and often regarded by certain data scientists as overkill.

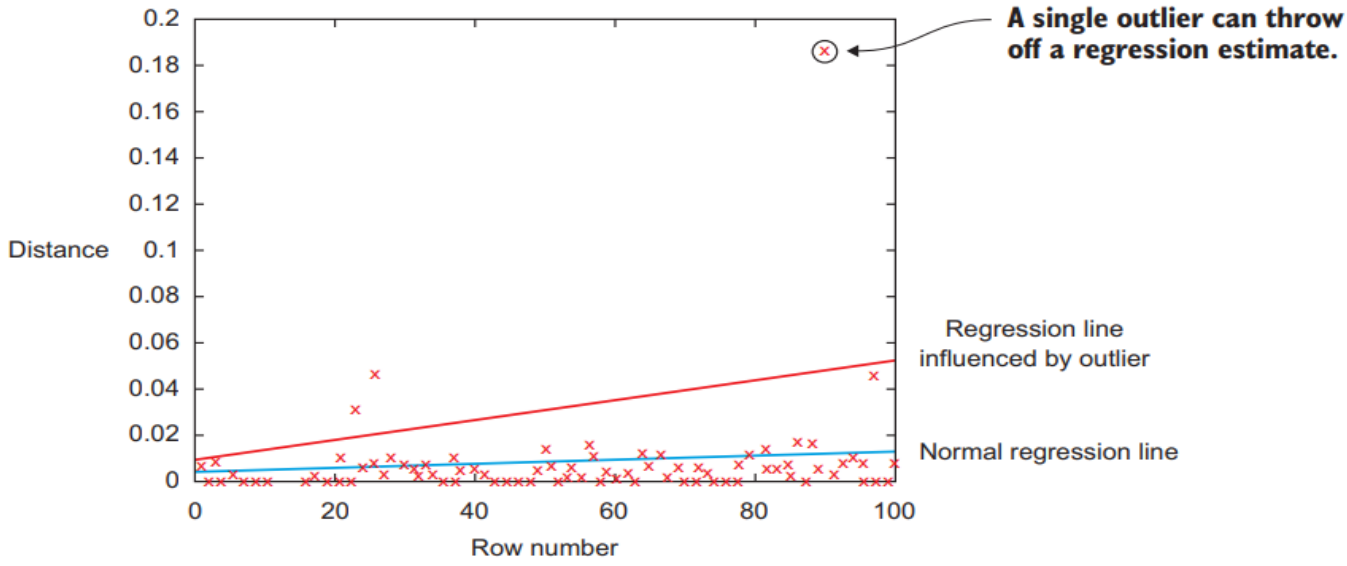


Figure 2.5 The encircled point influences the model heavily and is worth investigating because it can point to a region where you don't have enough data or might indicate an error in the data, but it also can be a valid data point.

- **DATA ENTRY ERRORS: -**

Data collection and data entry are error-prone processes. They often require human intervention, and because humans are only human, they make typos or lose their concentration for a second and introduce an error into the chain. But data collected by machines or computers isn't free from errors either. Errors can arise from human sloppiness, whereas others are due to machine or hardware failure. Examples of errors originating from machines are transmission errors or bugs in the **extract, transform, and load phase (ETL)**.

For small data sets you can check every value by hand. Detecting data errors when the variables you study don't have many classes can be done by tabulating the data with counts. When you have a variable that can take only two values: "Good" and "Bad", you can create a frequency table and see if those are truly the only two values present. In table 2.3, the values "Godo" and "Bade" point out something went wrong in at least 16 cases.

Table 2.3 Detecting outliers on simple variables with a frequency table

Value	Count
Good	1598647
Bad	1354468
Godo	15
Bade	1

Most errors of this type are easy to fix with simple assignment statements and if-thenelse rules:

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

- **REDUNDANT WHITESPACE: -**

  Whitespaces tend to be hard to detect but cause errors like other redundant characters would. Who hasn't lost a few days in a project because of a bug that was caused by whitespaces at the end of a string? You ask the program to join two keys and notice that observations are missing from the output file. After looking for days through the code, you finally find the bug. Then comes the hardest part: explaining the delay to the project stakeholders. The cleaning during the ETL phase wasn't well executed, and keys in one table contained a whitespace at the end of a string. This caused a mismatch of keys such as "FR " – "FR", dropping the observations that couldn't be matched.

  If you know to watch out for them, fixing redundant whitespaces is luckily easy enough in most programming languages. They all provide string functions that will remove the leading and trailing whitespaces. For instance, in Python you can use the strip() function to remove leading and trailing spaces.

  - **FIXING CAPITAL LETTER MISMATCHES: -** Capital letter mismatches are common. Most programming languages make a distinction between "Brazil" and "brazil". In this case you can solve the problem by applying a function that returns both strings in lowercase, such as .lower() in Python. "Brazil".lower() == "brazil".lower() should result in true.

- **IMPOSSIBLE VALUES AND SANITY CHECKS: -**

  Sanity checks are another valuable type of data check. Here you check the value against physically or theoretically impossible values such as people taller than 3 meters or someone with an age of 299 years. Sanity checks can be directly expressed with rules:

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

- **OUTLIERS: -**

  An outlier is an observation that seems to be distant from other observations or, more specifically, one observation that follows a different logic or generative process than the other observations. The easiest way to find outliers is to use a plot or a table with the minimum and maximum values. An example is shown in figure 2.6.

  The plot on the top shows no outliers, whereas the plot on the bottom shows possible outliers on the upper side when a normal distribution is expected. The **normal distribution**, or **Gaussian distribution**, is the most common distribution in natural sciences.

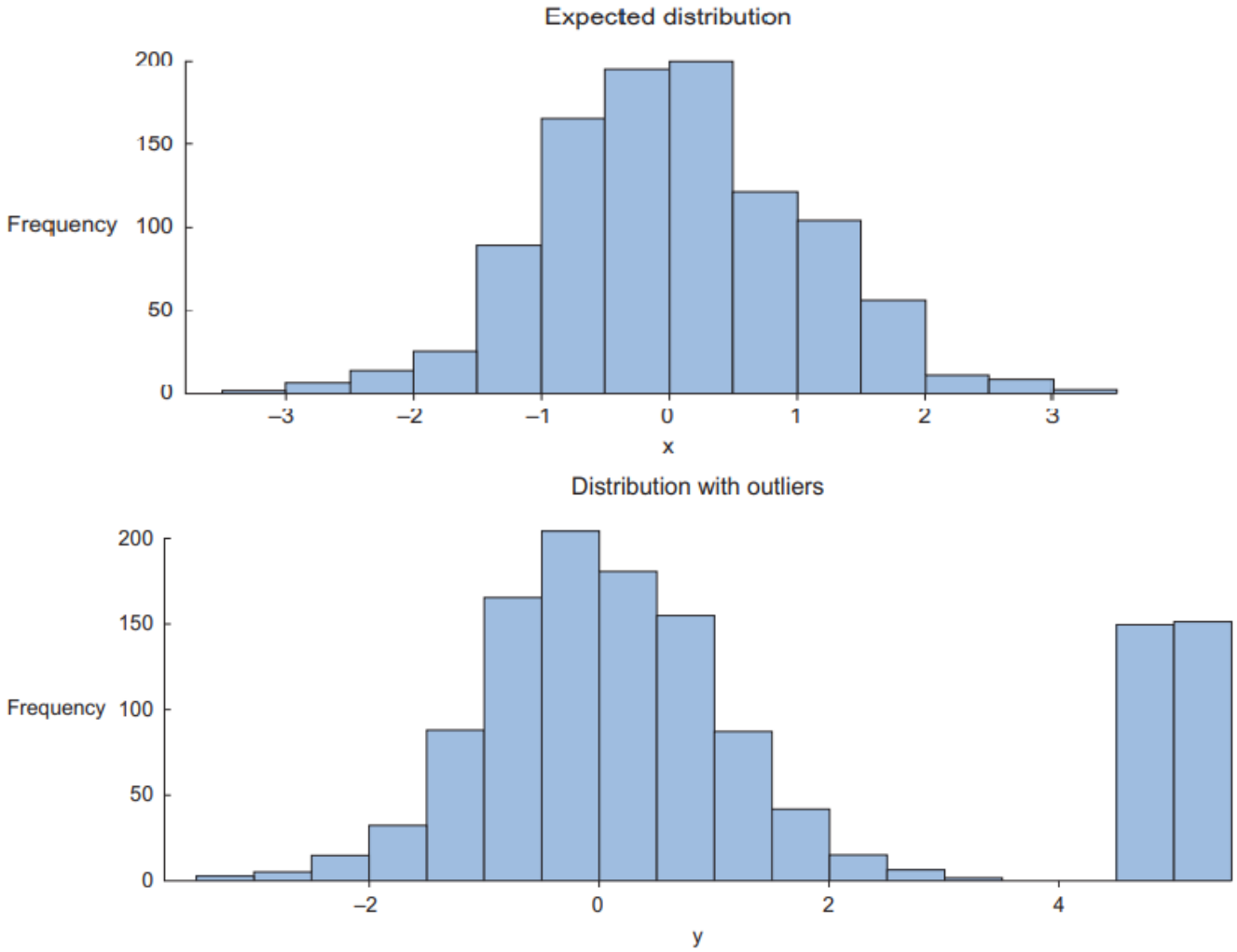


Figure 2.6 Distribution plots are helpful in detecting outliers and helping you understand the variable.

It shows most cases occurring around the average of the distribution and the occurrences decrease when further away from it. The high values in the bottom graph can point to outliers when assuming a normal distribution. As we saw earlier with the regression example, outliers can gravely influence your data modeling, so investigate them first.

- **DEALING WITH MISSING VALUES: -**

  Missing values aren't necessarily wrong, but you still need to handle them separately; certain modeling techniques can't handle missing values. They might be an indicator that something went wrong in your data collection or that an error happened in the ETL process. Common techniques data scientists use are listed in table 2.4.

Table 2.4 An overview of techniques to handle missing data

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

Which technique to use at what time is dependent on your particular case? If, for instance, you don't have observations to spare, omitting an observation is probably not an option. If the variable can be described by a stable distribution, you could impute based on this. However, maybe a missing value actually means "zero"? This can be the case in sales for instance: if no promotion is applied on a customer basket, that customer's promo is missing, but most likely it's also 0, no price cut.

- **DEVIATIONS FROM A CODE BOOK: -**

Detecting errors in larger data sets against a code book or against standardized values can be done with the help of set operations. A code book is a description of your data, a form of metadata. It contains things such as the number of variables per observation, the number of observations, and what each encoding within a variable means. (For instance "0" equals "negative", "5" stands for "very positive".) A code book also tells the type of data you're looking at: is it hierarchical, graph, something else?

You look at those values that are present in set A but not in set B. These are values that should be corrected. It's no coincidence that sets are the data structure that we'll use when we're working in code. It's a good habit to give your data structures additional thought; it can save work and improve the performance of your program.

If you have multiple values to check, it's better to put them from the code book into a table and use a difference operator to check the discrepancy between both tables.

- **DIFFERENT UNITS OF MEASUREMENT: -**

  When integrating two data sets, you have to pay attention to their respective units of measurement. An example of this would be when you study the prices of gasoline in the world. To do this you gather data from different data providers. Data sets can contain prices per gallon and others can contain prices per liter. A simple conversion will do the trick in this case.

- **DIFFERENT LEVELS OF AGGREGATION: -**

  Having different levels of aggregation is similar to having different types of measurement. An example of this would be a data set containing data per week versus one containing data per work week. This type of error is generally easy to detect, and summarizing (or the inverse, expanding) the data sets will fix it.

Correct errors as early as possible: -

A good practice is to mediate data errors as early as possible in the data collection chain and to fix as little as possible inside your program while fixing the origin of the problem. Retrieving data is a difficult task, and organizations spend millions of dollars on it in the hope of making better decisions. The data collection process is errorprone, and in a big organization it involves many steps and teams

Data should be cleansed when acquired for many reasons:

- Not everyone spots the data anomalies. Decision-makers may make costly mistakes on information based on incorrect data from applications that fail to correct for the faulty data.
- If errors are not corrected early on in the process, the cleansing will have to be done for every project that uses that data.
- Data errors may point to a business process that isn't working as designed. For instance, both authors worked at a retailer in the past, and they designed a couponing system to attract more people and make a higher profit. During a data science project, we discovered clients who abused the couponing system and earned money while purchasing groceries. The goal of the couponing system was to stimulate cross-selling, not to give products away for free. This flaw cost the company money and nobody in the company was aware of it. In this case the data wasn't technically wrong but came with unexpected results.
- Data errors may point to defective equipment, such as broken transmission lines and defective sensors.
- Data errors can point to bugs in software or in the integration of software that may be critical to the company. While doing a small project at a bank we discovered that two software applications used different local settings. This caused problems with numbers greater than 1,000. For one app the number 1.000 meant one, and for the other it meant one thousand.

Fixing the data as soon as it's captured is nice in a perfect world. Sadly, a data scientist doesn't always have a say in the data collection and simply telling the IT department to fix certain things may not make it so. If you can't correct the data at the source, you'll need to handle it inside your code. Data manipulation doesn't end with correcting mistakes; you still need to combine your incoming data.

As a final remark: always keep a copy of your original data (if possible). Sometimes you start cleaning data but you'll make mistakes: impute variables in the wrong way, delete outliers that had interesting additional information, or alter data as the result of an initial misinterpretation. If you keep a copy you get to try again. For "flowing data" that's manipulated at the time of arrival, this isn't always possible and you'll have accepted a period of tweaking before you get to use the data you are capturing. One of the more difficult things isn't the data cleansing of individual data sets however, it's combining different sources into a whole that makes more sense.

Combining data from different data sources: -

Your data comes from several different places, and in this sub step we focus on integrating these different sources. Data varies in size, type, and structure, ranging from databases and Excel files to text documents.

- **THE DIFFERENT WAYS OF COMBINING DATA: -**

  You can perform two operations to combine information from different data sets. The first operation is joining: enriching an observation from one table with information from another table. The second operation is appending or stacking: adding the observations of one table to those of another table.

  When you combine data, you have the option to create a new physical table or a virtual table by creating a view. The advantage of a view is that it doesn't consume more disk space.

- **JOINING TABLES: -**

  Joining tables allows you to combine the information of one observation found in one table with the information that you find in another table. The focus is on enriching a single observation. Let's say that the first table contains information about the purchases of a customer and the other table contains information about the region where your customer lives. Joining the tables allows you to combine the information so that you can use it for your model, as shown in figure 2.7.

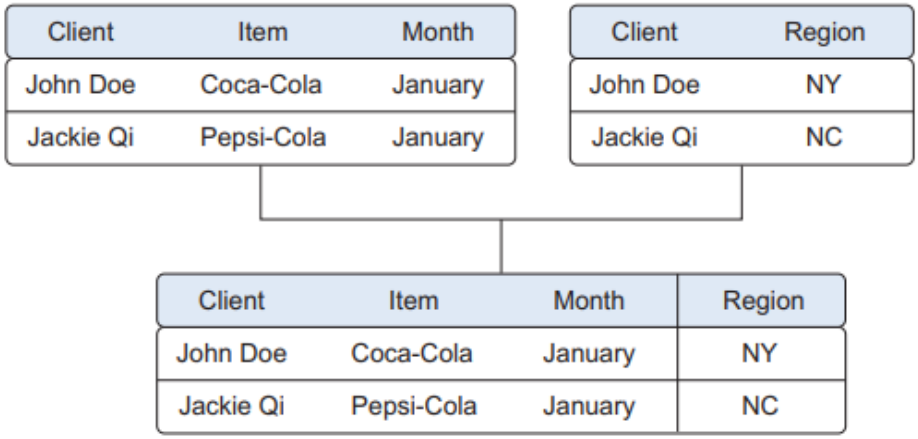


Figure 2.7 Joining two tables on the Item and Region keys

To join tables, you use variables that represent the same object in both tables, such as a date, a country name, or a Social Security number. These common fields are known as keys. When these keys also uniquely define the records in the table they are called primary keys.

One table may have buying behavior and the other table may have demographic information on a person. In figure 2.7 both tables contain the client name, and this makes it easy to enrich the client expenditures with the region of the client. People who are acquainted with Excel will notice the similarity with using a lookup function.

- **APPENDING TABLES: -**
  Appending or stacking tables is effectively adding observations from one table to another table. Figure 2.8 shows an example of appending tables. One table contains the observations from the month January and the second table contains observations from the month February. The result of appending these tables is a larger one with the observations from January as well as February. The equivalent operation in set theory would be the union, and this is also the command in SQL, the common language of relational databases. Other set operators are also used in data science, such as set difference and intersection.

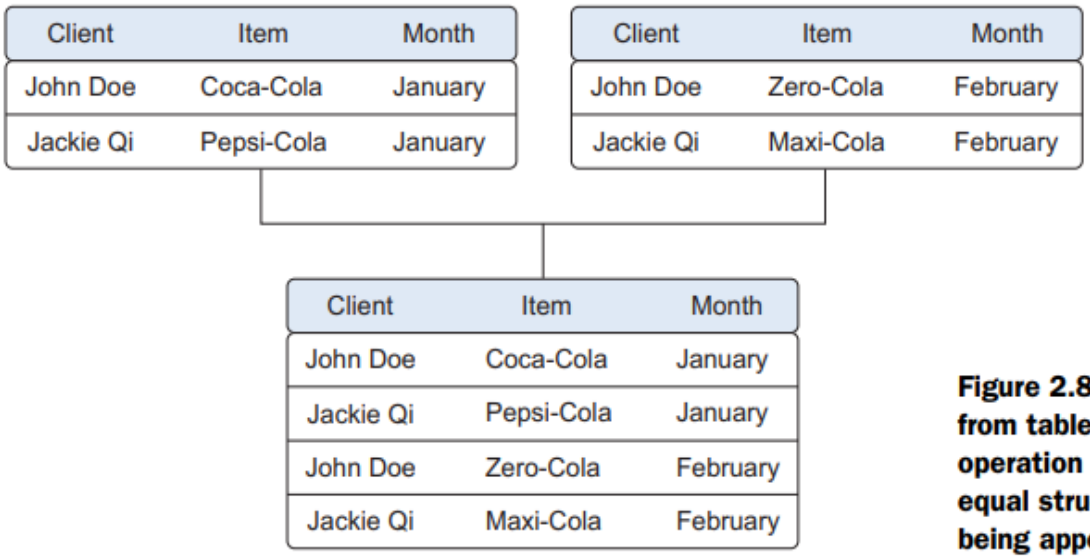


Figure 2.8 Appending data from tables is a common operation but requires an equal structure in the tables being appended.

- **USING VIEWS TO SIMULATE DATA JOINS AND APPENDS: -**
  To avoid duplication of data, you virtually combine data with views. In the previous example we took the monthly data and combined it in a new physical table. The problem is that we duplicated the data and therefore needed more storage space. In the example we're working with, that may not cause problems, but imagine that every table consists of terabytes of data; then it becomes problematic to duplicate the data. For this reason, the concept of a view was invented. A view behaves as if you're working on a table, but this table is nothing but a virtual layer that combines the tables for you. Figure 2.9 shows how the sales data from the different months is combined virtually into a yearly sales table instead of duplicating the data. Views do come with a drawback, however. While a table join is only performed once, the join that creates the view is recreated every time it's queried, using more processing power than a pre-calculated table would have.

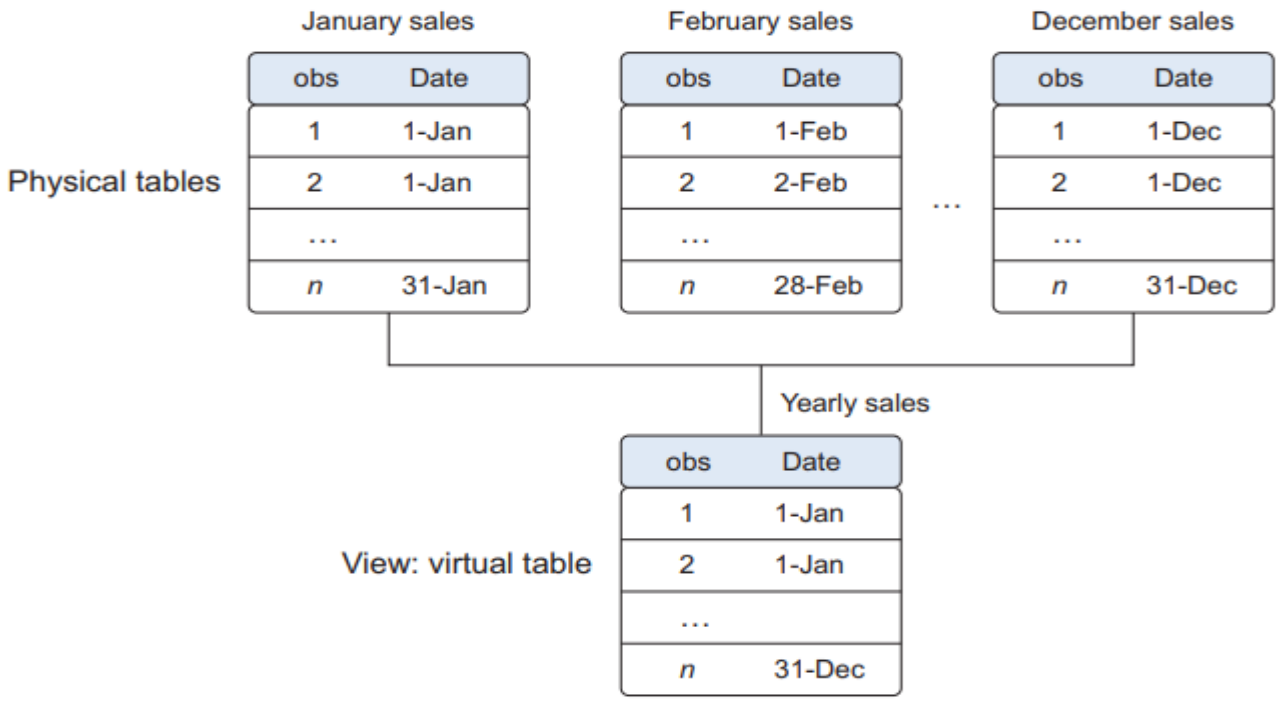


Figure 2.9 A view helps you combine data without replication.

- **ENRICHING AGGREGATED MEASURES: -**
  Data enrichment can also be done by adding calculated information to the table, such as the total number of sales or what percentage of total stock has been sold in a certain region (figure 2.10).
  Extra measures such as these can add perspective. Looking at figure 2.10, we now have an aggregated data set, which in turn can be used to calculate the participation of each product within its category. This could be useful during data exploration but more so when creating data models. As always this depends on the exact case, but from our experience models with "relative measures" such as % sales (quantity of product sold/total quantity sold) tend to outperform models that use the raw numbers (quantity sold) as input.

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

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

Transforming data: -

Certain models require their data to be in a certain shape. Now that you've cleansed and integrated the data, this is the next task you'll perform: transforming your data so it takes a suitable form for data modeling.

TRANSFORMING DATA: -

Relationships between an input variable and an output variable aren't always linear. Take, for instance, a relationship of the form $y = ae^{bx}$. Taking the log of the independent variables simplifies the estimation problem dramatically. Figure 2.11 shows how transforming the input variable greatly simplifies the estimation problem. Other times you might want to combine two variables into a new variable.

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

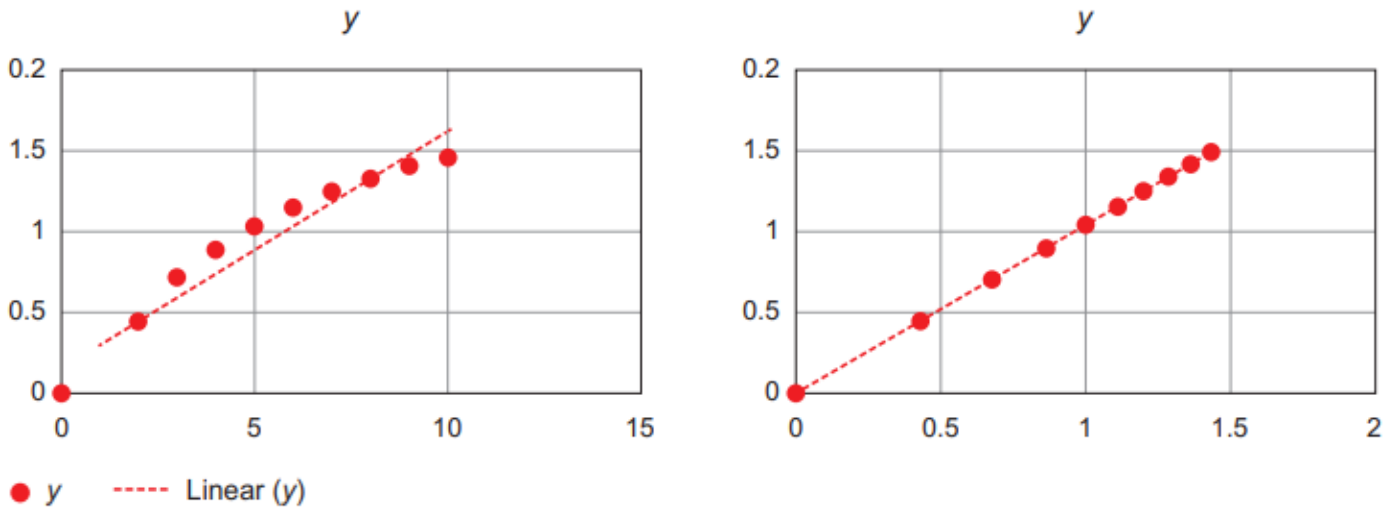


Figure 2.11 Transforming x to log x makes the relationship between x and y linear (right), compared with the non-log x (left).

REDUCING THE NUMBER OF VARIABLES: -

Sometimes you have too many variables and need to reduce the number because they don't add new information to the model. Having too many variables in your model makes the model difficult to handle, and certain techniques don't perform well when you overload them with too many input variables. For instance, all the techniques based on a Euclidean distance perform well only up to 10 variables.

Euclidean distance

Euclidean distance or "ordinary" distance is an extension to one of the first things anyone learns in mathematics about triangles (trigonometry): Pythagoras's leg theorem. If you know the length of the two sides next to the 90° angle of a right-angled triangle you can easily derive the length of the remaining side (hypotenuse). The formula for this is hypotenuse = $\sqrt{(side1 + side2)^2}$. The Euclidean distance between two points in a two-dimensional plane is calculated using a similar formula: distance = $\sqrt{((x1 - x2)^2 + (y1 - y2)^2)}$. If you want to expand this distance calculation to more dimensions, add the coordinates of the point within those higher dimensions to the formula. For three dimensions we get distance = $\sqrt{((x1 - x2)^2 + (y1 - y2)^2 + (z1 - z2)^2)}$.

Data scientists use special methods to reduce the number of variables but retain the maximum amount of data. Figure 2.12 shows how reducing the number of variables makes it easier to understand the key values. It also shows how two variables account for 50.6% of the variation within the data set (component1 = 27.8% + component2 = 22.8%). These variables, called "component1" and "component2," are both combinations of the original variables. They're the principal components of the underlying data structure.

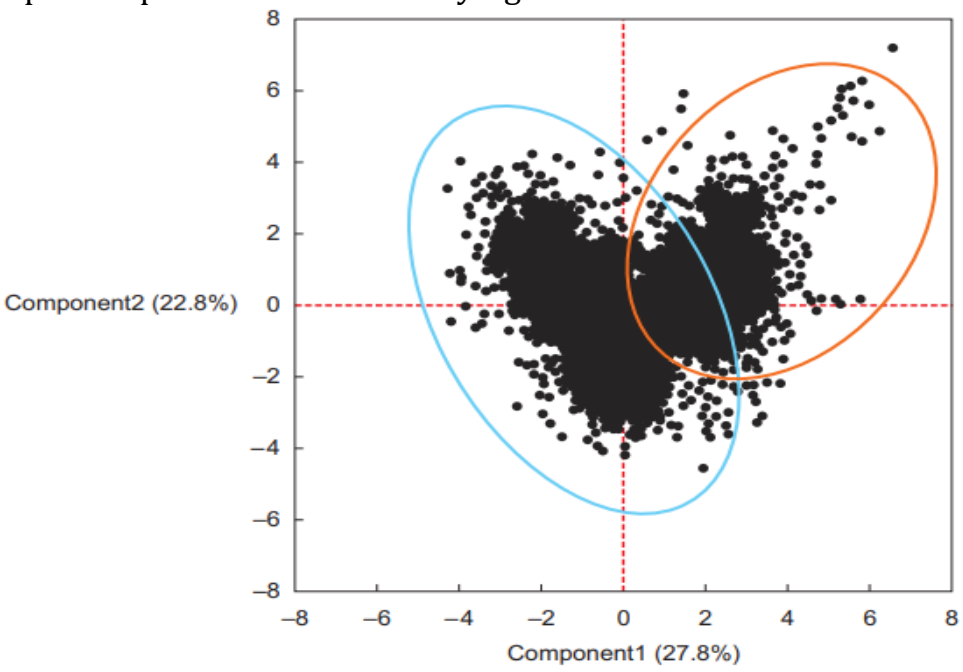


Figure 2.12 Variable reduction allows you to reduce the number of variables while maintaining as much information as possible.

TURNING VARIABLES INTO DUMMIES: -

Variables can be turned into dummy variables (figure 2.13). Dummy variables can only take two values: true(1) or false(0). They're used to indicate the absence of a categorical effect that may explain the observation. In this case you'll make separate columns for the classes stored in one variable and indicate it with 1 if the class is present and 0 otherwise. An example is turning one column named Weekdays into the columns Monday through Sunday. You use an indicator to show if the observation was on a Monday; you put 1 on Monday and 0 elsewhere.

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

M F

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

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

We introduced the third step in the data science process—cleaning, transforming, and integrating data—which changes your raw data into usable input for the modeling phase. The next step in the data science process is to get a better understanding of the content of the data and the relationships between the variables and observations.

Step 4: Exploratory data analysis: -

During exploratory data analysis you take a deep dive into the data (see figure 2.14). Information becomes much easier to grasp when shown in a picture, therefore you mainly use graphical techniques to gain an understanding of your data and the interactions between variables. This phase is about exploring data, so keeping your mind open and your eyes peeled is essential during the exploratory data analysis phase. The goal isn't to cleanse the data, but it's common that you'll still discover anomalies you missed before, forcing you to take a step back and fix them.

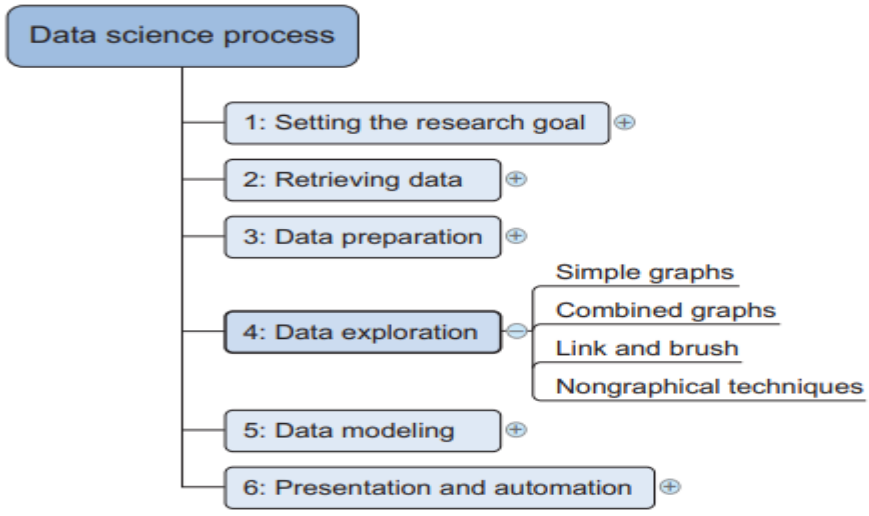


Figure 2.14 Step 4: Data exploration

The visualization techniques you use in this phase range from simple line graphs or histograms, as shown in figure 2.15, to more complex diagrams such as Sankey and network graphs. Sometimes it's useful to compose a composite graph from simple graphs to get even more insight into the data. Other times the graphs can be animated or made interactive to make it easier.

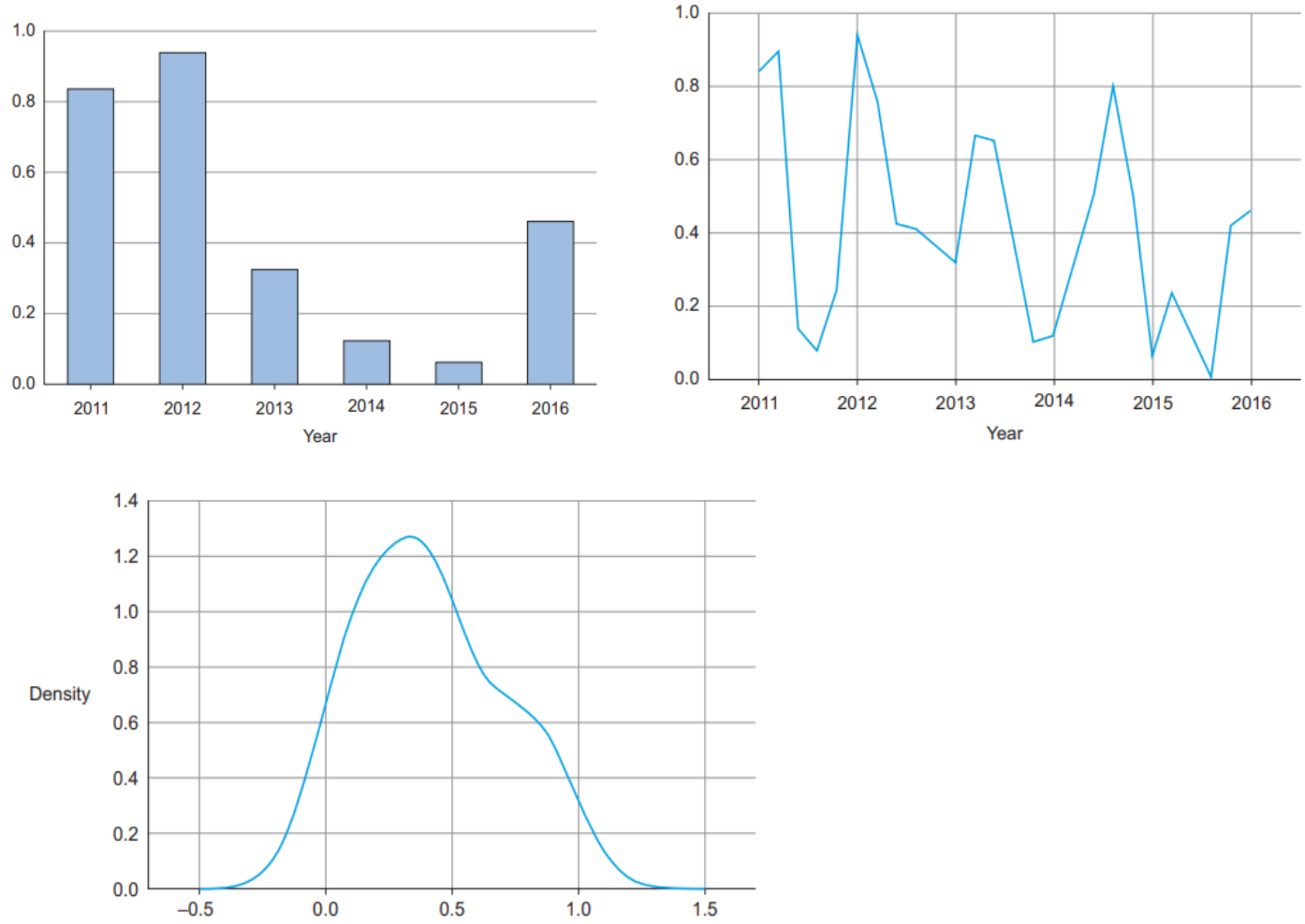
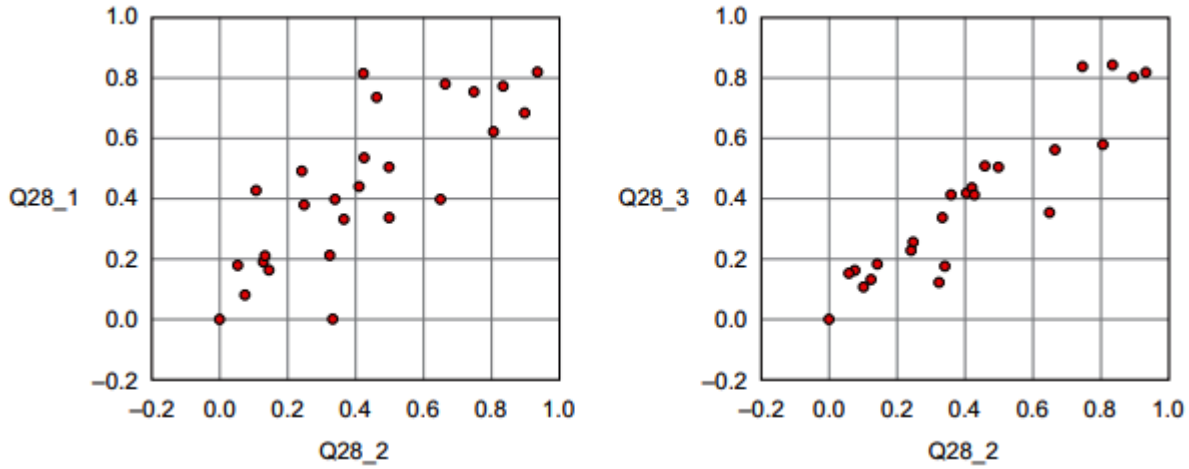


Figure 2.15 From top to bottom, a bar chart, a line plot, and a distribution are some of the graphs used in exploratory analysis.

These plots can be combined to provide even more insight, as shown in figure 2.16.



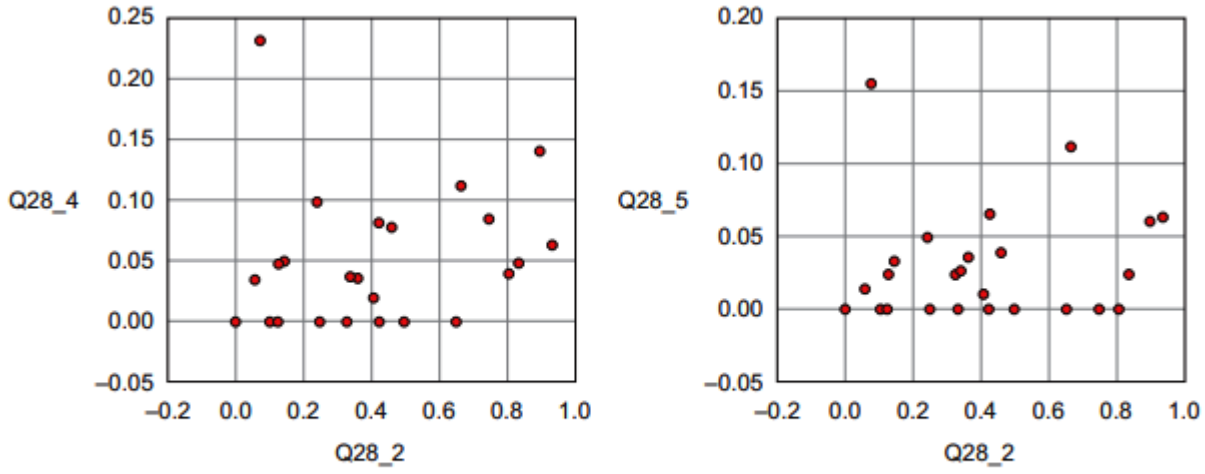


Figure 2.16 Drawing multiple plots together can help you understand the structure of your data over multiple variables.

Overlaying several plots is common practice. In figure 2.17 we combine simple graphs into a Pareto diagram, or 80-20 diagram.

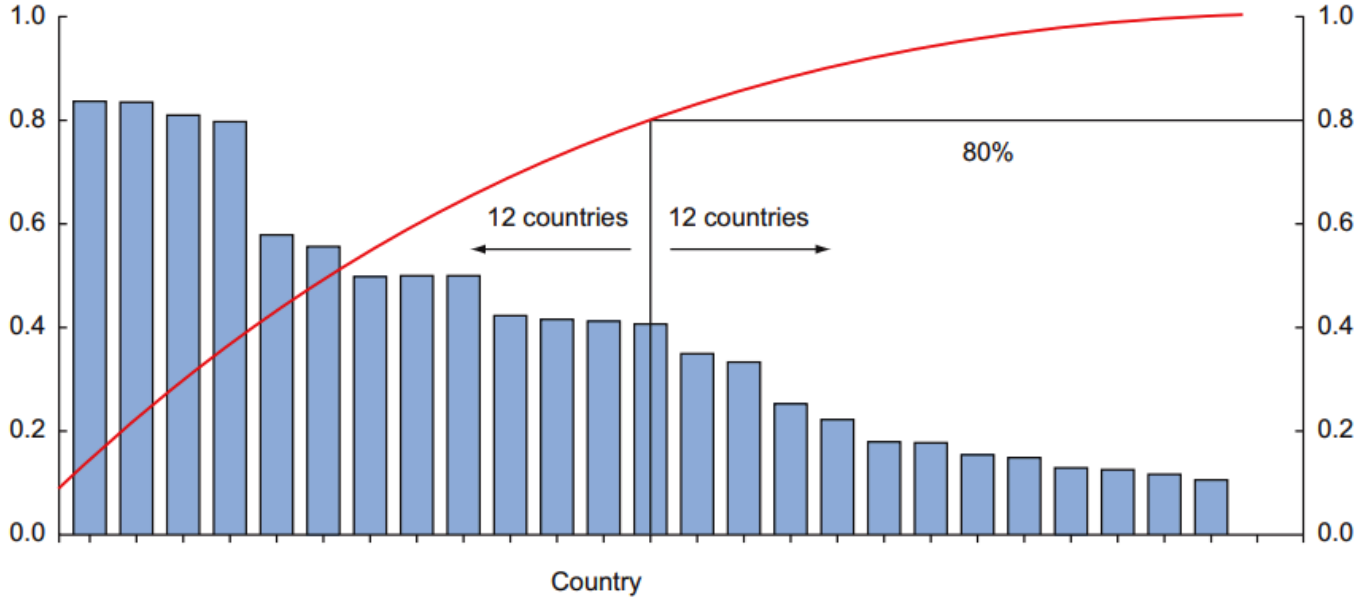


Figure 2.17 A Pareto diagram is a combination of the values and a cumulative distribution. It's easy to see from this diagram that the first 50% of the countries contain slightly less than 80% of the total amount. If this graph represented customer buying power and we sell expensive products, we probably don't need to spend our marketing budget in every country; we could start with the first 50%.

Figure 2.18 shows another technique: brushing and linking. With brushing and linking you combine and link different graphs and tables (or views) so changes in one graph are automatically transferred to the other graphs. This interactive exploration of data facilitates the discovery of new insights.

Figure 2.18 shows the average score per country for questions. Not only does this indicate a high correlation between the answers, but it's easy to see that when you select several points on a subplot, the points will correspond to similar points on the other graphs. In this case the

selected points on the left graph correspond to points on the middle and right graphs, although they correspond better in the middle and right graphs.

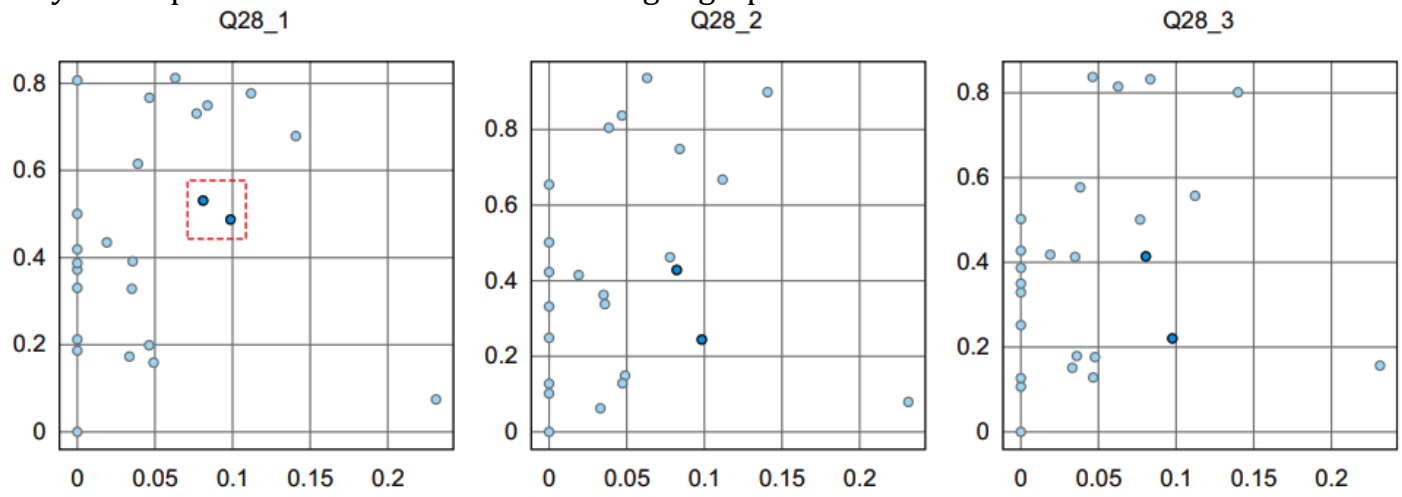


Figure 2.18 Link and brush allows you to select observations in one plot and highlight the same observations in the other plots.

Two other important graphs are the histogram shown in figure 2.19 and the boxplot shown in figure 2.20.

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

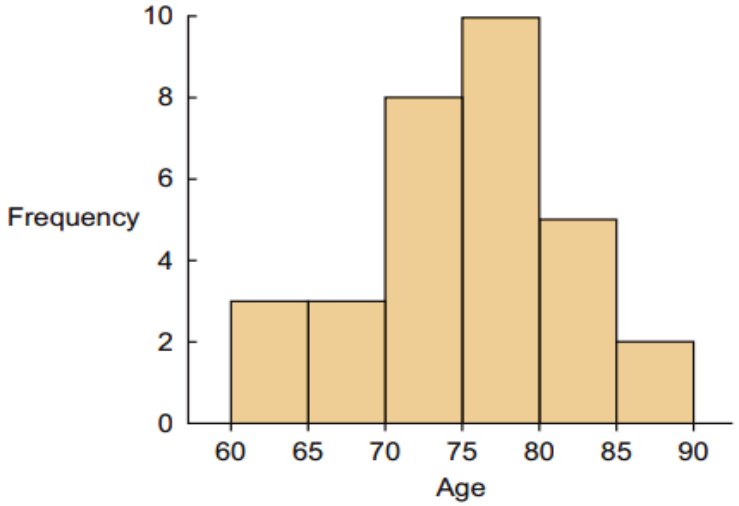


Figure 2.19 Example histogram: the number of people in the age-groups of 5-year intervals

The techniques we described in this phase are mainly visual, but in practice they're certainly not limited to visualization techniques. Tabulation, clustering, and other modeling techniques can also be a part of exploratory analysis. Even building simple models can be a part of this step.

Now that you've finished the data exploration phase and you've gained a good grasp of your data, it's time to move on to the next phase: building models.

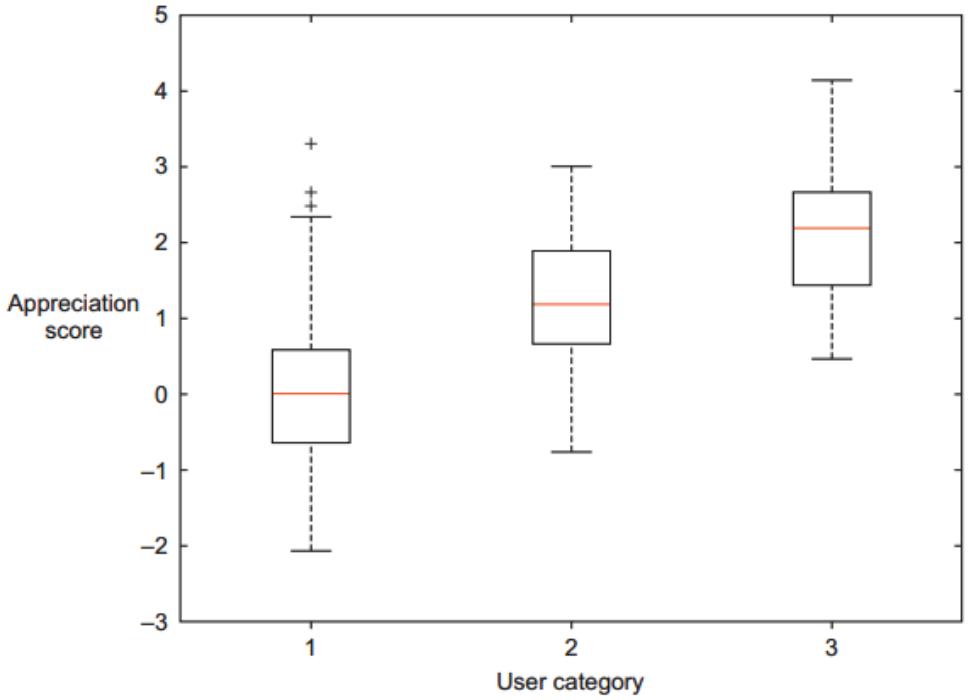


Figure 2.20 Example boxplot: each user category has a distribution of the appreciation each has for a certain picture on a photography website.

Step 5: Build the models: -

With clean data in place and a good understanding of the content, you're ready to build models with the goal of making better predictions, classifying objects, or gaining an understanding of the system that you're modeling. This phase is much more focused than the exploratory analysis step, because you know what you're looking for and what you want the outcome to be. Figure 2.21 shows the components of model building.

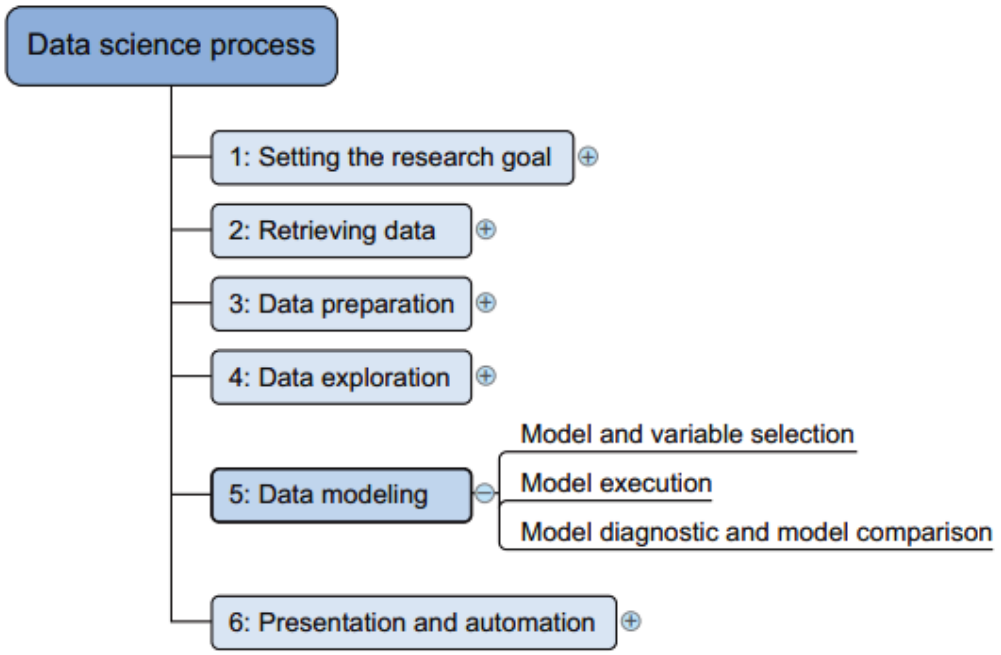


Figure 2.21 Step 5: Data modeling

The techniques you'll use now are borrowed from the field of machine learning, data mining, and/or statistics. Building a model is an iterative process. The way you build your model depends on whether you go with classic statistics or the somewhat more recent machine learning school, and the type of technique you want to use. Either way, most models consist of the following main steps:

1. Selection of a modeling technique and variables to enter in the model
2. Execution of the model
3. Diagnosis and model comparison

1. **Model and variable selection: -**
   You'll need to select the variables you want to include in your model and a modeling technique. Your findings from the exploratory analysis should already give a fair idea of what variables will help you construct a good model. Many modeling techniques are available, and choosing the right model for a problem requires judgment on your part. You'll need to consider model performance and whether your project meets all the requirements to use your model, as well as other factors:
   - Must the model be moved to a production environment and, if so, would it be easy to implement?
   - How difficult is the maintenance on the model: how long will it remain relevant if left untouched?
   - Does the model need to be easy to explain?

2. **Model execution: -**
   Once you've chosen a model you'll need to implement it in code. Luckily, most programming languages, such as Python, already have libraries such as StatsModels or Scikit-learn. These packages use several of the most popular techniques. Coding a model is a nontrivial task in most cases, so having these libraries available can speed up the process. As you can see in the following code, it's fairly easy to use linear regression (figure 2.22) with StatsModels or Scikit-learn. Doing this yourself would require much more effort even for the simple techniques.
   The following listing shows the execution of a linear prediction model.

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

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

Imports required Python modules.

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

Fits linear regression on data.

Shows model fit statistics.

For a linear regression, a "linear relation" between each x (predictor) and the y (target) variable is assumed, as shown in figure 2.22.

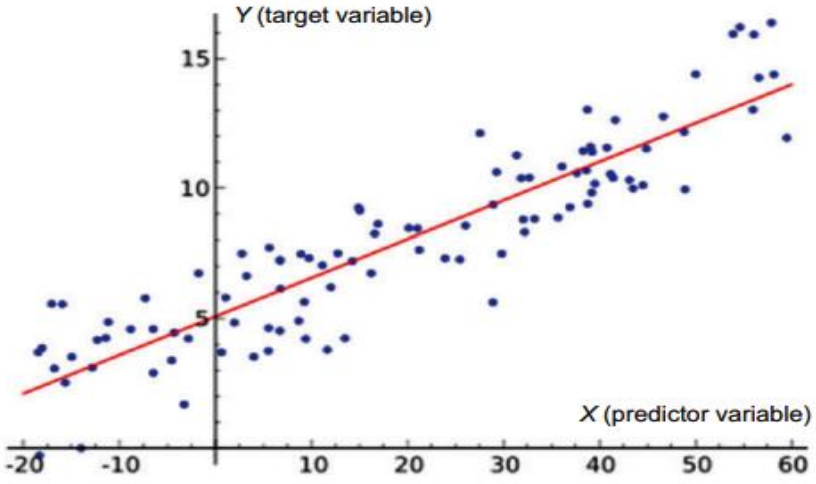


Figure 2.22 Linear regression tries to fit a line while minimizing the distance to each point

We, however, created the target variable, based on the predictor by adding a bit of randomness. It shouldn't come as a surprise that this gives us a well-fitting model. The results.summary() outputs the table in figure 2.23. Mind you, the exact outcome depends on the random variables you got.

Dep. Variable:	y	**R-squared:**	0.893
Model:	OLS	**Adj. R-squared:**	0.893
Method:	Least Squares	**F-statistic:**	2088.
Date:	Fri, 30 Oct 2015	**Prob (F-statistic):**	7.13e-243
Time:	12:44:31	**Log-Likelihood:**	-176.74
No. Observations:	500	**AIC:**	357.5
Df Residuals:	498	**BIC:**	365.9
Df Model:	2		
Covariance Type:	nonrobust		

Model fit: higher is better but too high is suspicious.

p-value to show whether a predictor variable has a significant influence on the target. Lower is better and <0.05 is often considered "significant."

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
x1	0.7658	0.040	19.130	0.000	0.687 0.844
x2	1.1252	0.039	28.603	0.000	1.048 1.202

Omnibus:	34.269	**Durbin-Watson:**	1.943
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	13.480
Skew:	-0.125	**Prob(JB):**	0.00118
Kurtosis:	2.235	**Cond. No.**	2.51

Linear equation coefficients.
y = 0.7658xl + l.l252x2.

Figure 2.23 Linear regression model information output

Let's ignore most of the output we got here and focus on the most important parts:

- **Model fit—** For this the R-squared or adjusted R-squared is used. This measure is an indication of the amount of variation in the data that gets captured by the model. The difference between the adjusted R-squared and the R-squared is minimal here because

the adjusted one is the normal one + a penalty for model complexity. A model gets complex when many variables (or features) are introduced. You don't need a complex model if a simple model is available, so the adjusted R-squared punishes you for overcomplicating. At any rate, 0.893 is high, and it should be because we cheated. Rules of thumb exist, but for models in businesses, models above 0.85 are often considered good. If you want to win a competition you need in the high 90s. For research however, often very low model fits (0.2 even) are found. What's more important there is the influence of the introduced predictor variables.

- **Predictor variables have a coefficient—** For a linear model this is easy to interpret. In our example if you add "1" to x1, it will change y by "0.7658". It's easy to see how finding a good predictor can be your route to a Nobel Prize even though your model as a whole is rubbish. If, for instance, you determine that a certain gene is significant as a cause for cancer, this is important knowledge, even if that gene in itself doesn't determine whether a person will get cancer. The example here is classification, not regression, but the point remains the same: detecting influences is more important in scientific studies than perfectly fitting models (not to mention more realistic). But when do we know a gene has that impact? This is called significance.
- **Predictor significance—**Coefficients are great, but sometimes not enough evidence exists to show that the influence is there. This is what the p-value is about. A long explanation about type 1 and type 2 mistakes is possible here but the short explanations would be: if the p-value is lower than 0.05, the variable is considered significant for most people. In truth, this is an arbitrary number. It means there's a 5% chance the predictor doesn't have any influence. Do you accept this 5% chance to be wrong? That's up to you. Several people introduced the extremely significant ($p<0.1$).

Linear regression works if you want to predict a value, but what if you want to classify something? Then you go to classification models, the best known among them being k-nearest neighbors.

As shown in figure 2.24, k-nearest neighbors looks at labeled points nearby an unlabeled point and, based on this, makes a prediction of what the label should be.

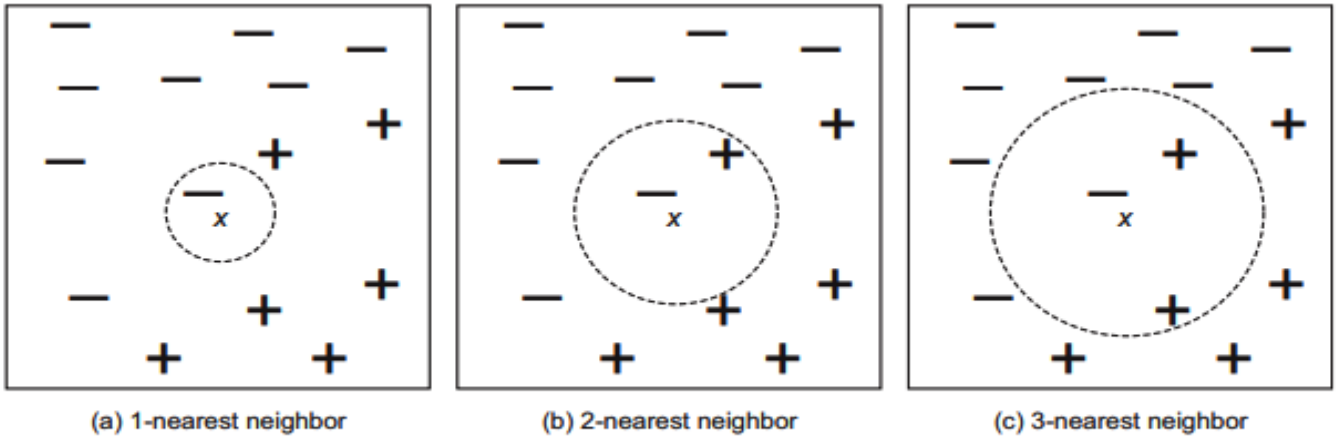


Figure 2.24 K-nearest neighbor techniques look at the k-nearest point to make a prediction.

Let's try it in Python code using the Scikit learn library, as in this next listing.

Listing 2.2 Executing k-nearest neighbor classification on semi-random data

```
from sklearn import neighbors
predictors = np.random.random(1000).reshape(500,2)
target = np.around(predictors.dot(np.array([0.4, 0.6])) +
         np.random.random(500))
clf = neighbors.KNeighborsClassifier(n_neighbors=10)
knn = clf.fit(predictors,target)
knn.score(predictors, target)
```

Imports modules.

Creates random predictor data and semi-random target data based on predictor data.

Fits 10-nearest neighbors model.

Gets model fit score: what percent of the classification was correct?

As before, we construct random correlated data and surprise, surprise we get 85% of cases correctly classified. If we want to look in depth, we need to score the model. Don't let knn.score() fool you; it returns the model accuracy, but by "scoring a model" we often mean applying it on data to make a prediction.

prediction = knn.predict(predictors)

Now we can use the prediction and compare it to the real thing using a confusion matrix.

metrics.confusion_matrix(target,prediction)

We get a 3-by-3 matrix as shown in figure 2.25.

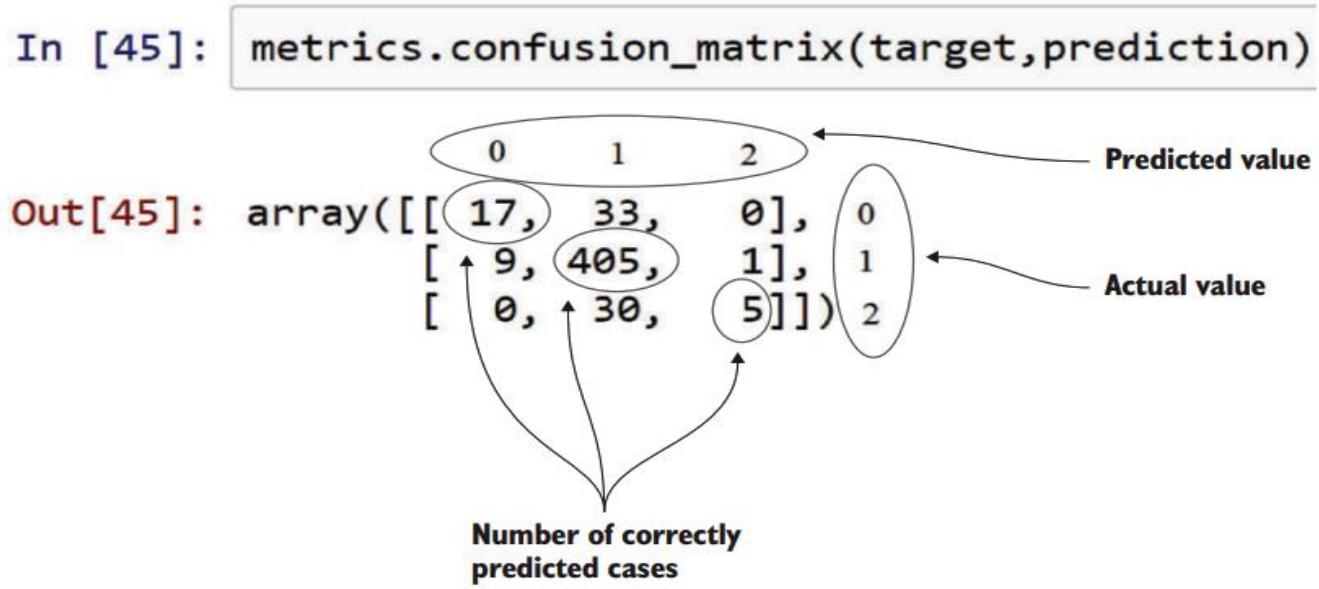


Figure 2.25 Confusion matrix: it shows how many cases were correctly classified and incorrectly classified by comparing the prediction with the real values. Remark: the classes (0,1,2) were added in the figure for clarification.

3. Model diagnostics and model comparison: -

You'll be building multiple models from which you then choose the best one based on multiple criteria. Working with a holdout sample helps you pick the best-performing model. A holdout sample is a part of the data you leave out of the model building so it can be used to evaluate the model afterward. The principle here is simple: the model should work on unseen data. You use only a fraction of your data to estimate the model and the other part, the holdout sample, is kept out of the equation. The model is then unleashed on the unseen data and error measures are calculated to evaluate it. Multiple error measures are available, and in figure 2.26 we show the general idea on comparing models. The error measure used in the example is the mean square error.

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

Figure 2.26 Formula for mean square error

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

Figure 2.27 compares the performance of two models to predict the order size from the price. The first model is size = 3 * price and the second model is size = 10. To estimate the models, we use 800 randomly chosen observations out of 1,000 (or 80%), without showing the other 20% of data to the model. Once the model is trained, we predict the values for the other 20% of the variables based on those for which we already know the true value, and calculate the model error with an error measure. Then we choose the model with the lowest error. In this example we chose model 1 because it has the lowest total error.

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

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

Many models make strong assumptions, such as independence of the inputs, and you have to verify that these assumptions are indeed met. This is called model diagnostics.

Step 6: Presenting findings and building applications on top of them

After you've successfully analyzed the data and built a well-performing model, you're ready to present your findings to the world (figure 2.28). This is an exciting part; all your hours of hard work have paid off and you can explain what you found to the stakeholders.

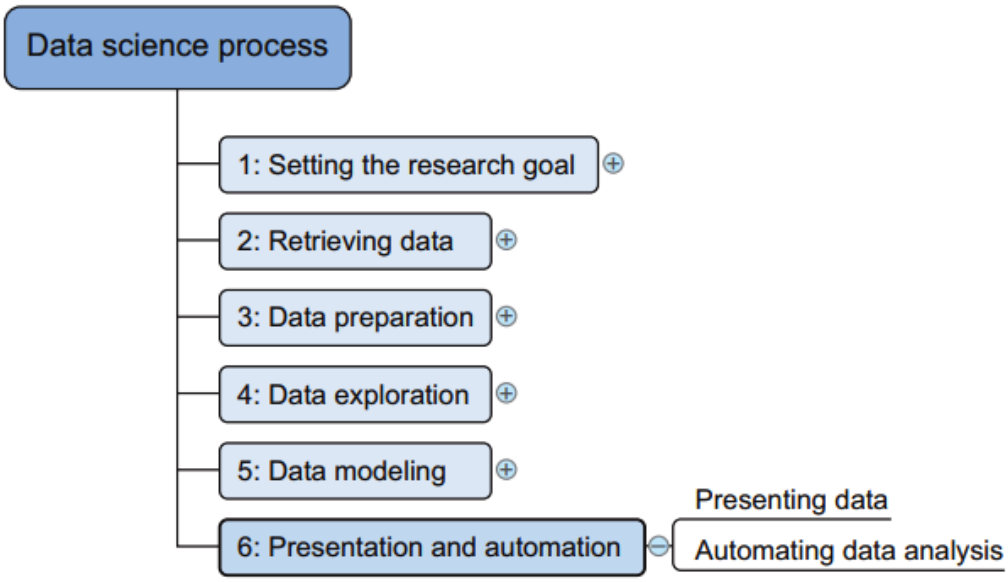


Figure 2.28 Step 6: Presentation and automation

Sometimes people get so excited about your work that you'll need to repeat it over and over again because they value the predictions of your models or the insights that you produced. For this reason, you need to automate your models. This doesn't always mean that you have to redo all of your analysis all the time. Sometimes it's sufficient that you implement only the model scoring; other times you might build an application that automatically updates reports, Excel spreadsheets, or PowerPoint presentations. The last stage of the data science process is where your soft skills will be most useful, and yes, they're extremely important.

UNIT WISE IMPORTANT QUESTIONS: -

1. What are the different types of errors and explain how to remove the errors?
2. Define error? Explain different types of errors.
3. Explain different types of error and with possible solutions
4. Explain different reasons why we should clean the data.
5. Explain how variables are turned into dummy variables
6. Explain different ways of combining data.
7. How can you transform your data so it takes a suitable form for data modeling?
8. Explain in detail about exploratory data analysis
9. What are the components of model building? Explain
10. Explain how model are compared in detail?
11. Explain in detail about model and variable selection
12. Explain different techniques to handle missing data along with advantages and disadvantages.