Data Analysis fundamentals

Learning Objectives

- Know what data analysis is
- Be able to do basic tasks related to data analysis
- Be able to write JavaScript code to consume our data
- Be able to use chartjs to display information about our data

Intro

Deriving insights from data enables us to make more informed decisions. We're currently in an era where we have unprecedented access to vast amounts of data. As a result, businesses are increasingly recognizing the value of harnessing data—turning to data analysis to uncover insights that drive their strategic objectives. The World Economic Forum's Future of Jobs Report 2025 identified data analysts and scientists as one of the fastest-growing professions, alongside AI and machine learning experts and big data specialists

A data analyst collects information, identifies meaningful patterns within it, and communicates their findings to colleagues to support decision-making. For example, as a data analyst in a healthcare setting, you might compile patient data to assess the effectiveness of specific treatments. In a marketing context, you could examine website visitor demographics to determine if your company is reaching its intended audience. Working as a data analyst in the financial industry, you might evaluate a company's investment portfolio to identify trends and provide guidance on future investment strategies.

What are the 5 steps of data analysis?

Data analysts use data to solve problems. As such, the data analysis process typically moves through several iterative phases. Let’s take a closer look at each:

1. Identify the business question you’d like to answer. What problem is the company trying to solve? What do you need to measure, and how will you measure it?

2. Collect the raw data sets you’ll need to help you answer the identified question. Data collection might come from internal sources, like a company’s client relationship management (CRM) software, or from secondary sources, like government records or social media application programming interfaces (APIs).

3. Clean the data to prepare it for analysis. This often involves purging duplicate and anomalous data, reconciling inconsistencies, standardizing data structure and format, and dealing with white spaces and other syntax errors.

4. Analyze the data. By manipulating the data using various data analysis techniques and tools, you can begin to find trends, correlations, outliers, and variations that tell a story. During this stage, you might use data mining to discover patterns within databases or data visualization software to help transform data into an easy-to-understand graphical format.

5. Interpret the results of your analysis to see how well the data answered your original question. What recommendations can you make based on the data? What are the limitations of your conclusions?

And if were ever wondering what's the difference between data analysis and analyitics, we got you covered! In practice, the terms are often used interchangeably, especially in job titles and descriptions. However, data analytics generally implies a more comprehensive and advanced approach to working with data.

Both data analysis and data analytics aim to extract valuable insights from data to inform decision-making and drive business value. The choice between the two terms often depends on the specific context, the complexity of the tasks involved, and the organizational preference.

Analysing our recipes

Let's get started with the first step! What is our business question?

I got a couple in my mind, but let's go with something that could look cool in our app. I'm interested in the ratio of recipes from different cuisines. Which one is the most popular, essentially?

There are different chart and graph types that could help here, but my aim is to make sure we get something that looks great and visually is obvious. One of the better choices is a pie or donut chart! We will look into how to implement one later.

The second step is gathering data. Luckily we already have a couple of recipes, but if we want our app to look cool, we can cheat a bit here. Either we can keep creating new recipes using our front end, or we can ask our chatbot buddies to generate identically structured recipes in a JSON format. Once we got that, we can copy and paste it into our app's recipes.json!

Next, the task is to clean the data. This might not be important as long as we are fully aware of what our recipes look like, but in a real scenario, this is extremely cruical - can't create reliable datasets if some of my recipes have empty values for cuisine - or worse, some gibberish.

Think of a different example: You could try to calculate either the average time it takes to make our recipes, or find the shortest/longest ones. If our data includes a ton of different measurements (15 minutes, 2 hours, 2 days) our analysis would be unreliable at best. So we have to change both the data either in our database or after reading it in, and we also would need to change the way we gather the new recipes, by making sure everything gets stored as a number.

So far, easy, right?

Do not worry, we will get to coding soon!

Next up, we need to actually analyse the data, then present it. Presenting it will be at the very end, first, let's figure out how we can gather the info needed.

We have 3 different options:

- Google the question, and with research, find the most optimal solution for our problems
- Write some pseudocode, figure out the solution ourselves
- Ask a chatbot model.

I am a huge fan of learning things on our own, but in the interest of time, we will not shy away from asking our studybuddy.

Let's see what the Claude 3.5 model has to say!

First we will try to phrase our problem in an easy to convey way (we can use this as a google phrase as well):

We would like to count the occurances of a value in a given JSON array of objects. We need the values and their occurances as well.

let's see what our chatbot has to say to a question like this:

```
// Parse the JSON array
const data = JSON.parse(recipes);

// Use reduce to count occurrences
const counts = data.reduce((accumulator, recipe) => {
    if (recipe.hasOwnProperty("cuisine")) {
        const cuisine = recipe.cuisine;
        if(accumulator.cuisine){
            accumulator.cuisine = accumulator.cuisine + 1;
        } else {
            accumulator.cuisine = 1;
        }
    }
    return accumulator;
}, {});
```

Now this is a doozy. (might be worth asking the chatbot about reduce() if you're unsure after the lesson!)

Essentially the job of reduce is to, well, *reduce* a large set of values to a single value. In our case, an array of objects to a new object, which counts occurances.

In short, the way the code works is like so:

- Get the data from our JSON file
- Using `reduce()`, loop through our data, while creating an accumulator that will work as the result of our reduce
- if the recipe we currently iterating through has a cuisine that we've seen before in the accumulator, increase the corresponding counter
- if it didn't exist before, add the cuisine to the accumulator with a counter of 1.

We can use a console.log() to check out the result before we do anything with it. Once we confirm visually that it looks good, we can start thinking about how can we use this data!

It'd be pretty easy to convert it to a JSON object, and send it over via a new route from our back end.

```
app.get("/cuisine-data", (req, res) => {
  const counts = data.reduce((accumulator, recipe) => {
  if (recipe.hasOwnProperty("cuisine")) {
      const cuisine = recipe.cuisine;
      if(accumulator.cuisine){
          accumulator.cuisine = accumulator.cuisine + 1;
      } else {
          accumulator.cuisine = 1;
      }
  }
  return accumulator;
  }, {});
  res.json(counts)
});
```

Once we confirm the data exists (we can call http://localhost:3000/cuisine-data from the browser), we can start thinking about how are we going to utilise this!

Data Visualisation

Data visualization is the graphical representation of information and data. It involves transforming complex data sets into visual formats like charts, graphs, maps, and other graphical elements. The main goal of data visualization is to make data more accessible, understandable, and actionable.

Key aspects of data visualization include:

1. Clarity: Presenting data in a clear, easy-to-understand format.
2. Insight: Revealing patterns, trends, and relationships within data that might not be apparent in raw form.
3. Communication: Effectively conveying information to a wide range of audiences, from experts to general viewers.
4. Decision-making: Supporting data-driven decision-making by making complex information more digestible.

Common types of data visualizations include bar charts, line graphs, pie charts, scatter plots, heat maps, and infographics. Each type serves different purposes and is suited to different kinds of data.

The process of creating effective data visualizations involves:

1. Understanding the data and its context
2. Choosing the appropriate visualization type
3. Designing the visual elements (colors, labels, scales)
4. Ensuring the visualization accurately represents the data without distortion

With the rise of big data, data visualization has become increasingly important in fields such as business intelligence, scientific research, and public communication. It allows us to process large amounts of information quickly and extract meaningful insights.

Modern data visualization often leverages software tools and programming libraries to create interactive and dynamic visualizations, enabling users to explore data in real-time and from multiple perspectives.

But what libraries are available in JavaScript?

ChartJS

One of the biggest charting tools is ChartJs, I've been personally using it for the better part of a decade. After looking around online, we can see the following sample code (as long as we have the necessary setup in the tag):

```
const xValues = ["Italy", "France", "Spain", "USA", "Argentina"];
const yValues = [55, 49, 44, 24, 15];

new Chart("myChart", {
  type: "pie",
  data: {
    labels: xValues,
    datasets: [{
      data: yValues
    }]
  },
  options: {
    title: {
      display: true,
      text: "World Wide Wine Production 2018"
    }
  }
});
```

This example shows us how can we utlise JS code to get displayed in a nice fashion.

Let's rework it with our code in our head:

```
const xValues = Object.keys(counts);
const yValues = Object.values(counts);

// optionally we can use colours here

new Chart("myChart", {
  type: "pie",
  data: {
    labels: xValues,
    datasets: [{
      data: yValues
    }]
```

```
    },
    options: {
      title: {
        display: true,
        text: "Cuisine Popularity"
      }
    }
  });
```

Easy as pie (ha!)

Let's write our front end code to consume this data.

Before we write any JavaScript, we need 2 things: Importing the Chart tool in the head tag, and creating a new

HTML element called a

in our `index.html`:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="style.css">
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Pacifico&display=swap"
rel="stylesheet">
  <script src="app.js" defer></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script> <!-- NEW -->
  <title>InstantGramen</title>
</head>

<body>
  <header>
   <!-- same as before -->
  <main>
    <h2>My Favourite Recipes</h2>
    <canvas id="myChart" style="width:100%;max-width:600px"></canvas>  <!-- NEW --
>
    <form>
      <p>Add new recipe:</p>
      <div class="form-row">
        <label for="name">Name: </label>
```

```
        <input type="text" name="name" id="name">
      </div>

    </ul>
  </main>
</body>

</html>
```

Now we can create our JS code at the bottom of our app.js file (make sure it's still inside the eventListener!):

```
const response = await fetch("http://localhost:3000/cuisine-data");
const cuisineData = await response.json();

const xValues = Object.keys(cuisineData);
const yValues = Object.values(cuisineData);

// optionally we can use colours here

new Chart("myChart", {
  type: "pie",
  data: {
    labels: xValues,
    datasets: [{
      data: yValues
    }]
  },
  options: {
    title: {
      display: true,
      text: "Cuisine Popularity"
    }
  }
});
```

Fantastic work! Now try adding more recipes to see the ratios change on the chart!