ECE2720: Data Science for Engineers

Instructor: Jayadev Acharya, Cornell

Notes by Joyce Shen, FA24

1 Chapter 1: A Primer on Python

1.1 Introduction to Python

- Python is a versatile, interpreted language, ideal for prototyping and data science.
- It is a high-level language that abstracts away many complexities (e.g., memory management).
- We primarily use Python 3.x for this course, as it has better support for modern applications, including numerical calculations and data visualization.

1.2 Basic Python Syntax

- Python can be used as a simple calculator:

```
¿¿¿ 1 + 1
2
¿¿¿ 5 / 3  # Python 3 automatically handles floating point division
1.6666666666666667
```

- Variables are dynamically typed and do not need to be declared before being used:

```
¿¿¿ x = 3
¿¿¿ y = 5
¿¿¿ z = y / x
¿¿¿ print(z)
1.6666666666666667
```

- Use `type()` to determine the type of a variable.

1.3 Basic Data Types

- **int**: Integer type, e.g., `x = 5`.

- **float**: Floating-point number, e.g., `y = 5.0`.
- **str**: Strings, e.g., `word = ”hello”`.
- **list**: Ordered sequences of elements, e.g., `lst = [1, 2, 3]`.
- **tuple**: Immutable sequences, e.g., `tup = (1, 2, 3)`.
- **dict**: Key-value pairs, e.g., `d = {’a’: 1, ’b’: 2}`.

1.4 Lists in Python

- Lists are a versatile data structure in Python, capable of storing different data types in the same sequence.
- Lists are mutable, meaning you can modify them in place.
- List indexing starts at 0:

```
¿¿¿ lst = [3, 4, 5, 6, 7]
¿¿¿ lst[0]
3
¿¿¿ lst[3]
6
¿¿¿ lst[4] = 8  # Modifying a list element
¿¿¿ print(lst)
[3, 4, 5, 6, 8]
```

- Slicing lists:

```
¿¿¿ lst[0:2]  # From index 0 to index 2 (exclusive)
[3, 4]
¿¿¿ lst[::-1]  # Reverse the list
[8, 6, 5, 4, 3]
```

1.5 Tuples and Strings

- **Tuples** are immutable sequences, meaning they cannot be modified in place:

```
¿¿¿ tup = (’hello’, 5, ’world’)
¿¿¿ print(tup)
(’hello’, 5, ’world’)
```

- Strings are sequences of characters and are immutable, like tuples:

```
¿¿¿ s = ’hello’
¿¿¿ s[0:3]
’hel’
¿¿¿ s + ’ world’
’hello world’
```

1.6 Control Structures

- Python uses indentation to define blocks of code.
- **If-then-else statements**:

```
¿¿¿ a = 1
¿¿¿ if a ¡ 0:
...     print('Negative')
... elif a == 0:
...     print('Zero')
... else:
...     print('Positive')
Positive
```

- **For loops**: Loops iterate over lists, strings, tuples, or any sequence:

```
¿¿¿ for i in range(4):
...     print(i ** 2)
0
1
4
9
```

1.7 List Comprehensions

- List comprehensions provide a concise way to create lists:

```
¿¿¿ squares = [x**2 for x in range(5)]
¿¿¿ print(squares)
[0, 1, 4, 9, 16]
```

1.8 Numpy: Numerical Python

- Numpy is a powerful library for numerical calculations, supporting large, multi-dimensional arrays.
- Create arrays:

```
¿¿¿ import numpy as np
¿¿¿ arr = np.array([2, 3, 4])
¿¿¿ print(arr)
[2 3 4]
```

- Numpy provides vectorized operations, allowing for efficient mathematical operations on arrays without loops.
- Create arrays of zeros, ones, or ranges of values:

```
¿¿¿ np.zeros(3)
array([0., 0., 0.])
¿¿¿ np.arange(5)
array([0, 1, 2, 3, 4])
¿¿¿ np.linspace(0, 1, 10)
array([0. , 0.1, 0.2, ..., 1.0])
```

1.9 Matplotlib: Plotting in Python

- Matplotlib provides plotting functionality similar to Matlab.
- Basic plot creation:

```
¿¿¿ import matplotlib.pyplot as plt
¿¿¿ x = np.linspace(0, 1, 100)
¿¿¿ y = x ** 2
¿¿¿ plt.plot(x, y)
¿¿¿ plt.xlabel('x')
¿¿¿ plt.ylabel('y')
¿¿¿ plt.title('y = x2')
¿¿¿ plt.show()
```

- Use `savefig` to save the plot to a file:

```
¿¿¿ plt.savefig('plot.pdf')
```

2 Numerical Optimization

Optimization problems are central to many areas in data science. At a high level, they involve finding the input x that makes a function $f(x)$ as small (or as large) as possible. These problems arise across fields such as machine learning, network design, and circuit optimization. Mathematically, the goal is to find $\vec{x}^*$ that minimizes $f(\vec{x})$ for a given function $f : \mathbb{R}^n \to \mathbb{R}$, where the domain $\mathbb{R}^n$ can be high-dimensional.

2.1 Global and Local Minima

A **global minimum** of a function f is defined as a point $\vec{x}^*$ such that:

$$f(\vec{x}^*) \leq f(\vec{x}) \quad \forall \vec{x}.$$

However, not all functions have global minima. For instance, the function $f(x) = -x^2$ does not have a global minimum because $f(x) \to -\infty$ as $x \to \infty$. We say that this function is not **bounded from below**. Functions that are bounded from below, like $f(x) = e^{-x}$, also don't necessarily have global minima, but the values of $f(x)$ are constrained to remain above a certain level (in this case, 0).

On the other hand, a **local minimum** is a point $\vec{x}^*$ such that for some small $\epsilon > 0$, $f(\vec{x}^*) \leq f(\vec{x})$ for all $\vec{x}$ in a neighborhood of $\vec{x}^*$ (i.e., points within ϵ distance).

2.2 Iterative Optimization Schemes

In practice, it's common to approximate the solution by starting from an initial guess $\vec{x}_1$ and iteratively improving it. The general strategy is to compute the function f at $\vec{x}_1$, move in the direction that reduces f, and repeat the process at $\vec{x}_2$. This is known as an **iterative method**.

Iterative methods are especially useful when n (the dimension of the domain) is large. A brute-force grid search for large n becomes infeasible since evaluating f on every possible point requires an exponential number of queries (e.g., 100^n for just 100 grid points in each direction).

2.3 Gradient Descent

Gradient descent is one of the most widely used methods for optimization. Given the function f, we compute its gradient, which points in the direction of the steepest ascent. To minimize the function, we move in the opposite direction:

$$\vec{x}_{n+1} = \vec{x}_n - \alpha \nabla f(\vec{x}_n),$$

where α is the step size, and $\nabla f(\vec{x}_n)$ is the gradient at $\vec{x}_n$. The choice of α is critical: too large a value might cause overshooting (missing the minimum), while too small a value might result in slow convergence.

2.4 Gradient and Hessian

For a function $f : \mathbb{R}^n \to \mathbb{R}$, the **gradient** $\nabla f(\vec{x})$ is a vector that gives the direction of the steepest increase at $\vec{x}$:

$$\nabla f(\vec{x}) = \left(\frac{\partial f(\vec{x})}{\partial x_1}, \frac{\partial f(\vec{x})}{\partial x_2}, \ldots, \frac{\partial f(\vec{x})}{\partial x_n} \right).$$

The **Hessian matrix** $\nabla^2 f(\vec{x})$ contains all second-order partial derivatives:

$$\nabla^2 f(\vec{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

The Hessian helps determine the curvature of f, and in many optimization algorithms, it's used to guide step size decisions.

2.5 Newton's Method

Newton's method is an iterative optimization method that uses both the gradient and Hessian to update the current guess $\vec{x}_n$:

$$\vec{x}_{n+1} = \vec{x}_n - \left(\nabla^2 f(\vec{x}_n)\right)^{-1} \nabla f(\vec{x}_n).$$

In this method, the Hessian provides a more precise estimate of how the function behaves locally, leading to faster convergence compared to basic gradient descent. However, computing and inverting the Hessian matrix can be computationally expensive, particularly for high-dimensional problems.

2.6 Convexity and Second Derivative Test

A function f is **convex** if for any two points $\vec{x}_1$ and $\vec{x}_2$ in its domain, and any $\theta \in [0, 1]$:

$$f(\theta\vec{x}_1 + (1-\theta)\vec{x}_2) \leq \theta f(\vec{x}_1) + (1-\theta) f(\vec{x}_2).$$

Graphically, this means the line segment connecting any two points on the function lies above the function itself.

Convex functions are important in optimization because any local minimum is guaranteed to be a global minimum. A quick way to check convexity for a twice-differentiable function is to examine the Hessian matrix:

$$f \text{ is convex if } \nabla^2 f(\vec{x}) \text{ is positive semi-definite for all } \vec{x}.$$

In one dimension, this reduces to checking that $f''(x) \geq 0$ for all x.

2.7 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a variation of gradient descent where, instead of computing the full gradient $\nabla f(\vec{x})$, a randomly chosen subset of the data (or function components) is used to estimate the gradient. This makes the method particularly efficient for large datasets. Formally, instead of computing:

$$\vec{x}_{n+1} = \vec{x}_n - \alpha \nabla f(\vec{x}_n),$$

SGD approximates the gradient:

$$\vec{x}_{n+1} = \vec{x}_n - \alpha \nabla f_i(\vec{x}_n),$$

where f_i is a randomly chosen function from the sum $f = \sum f_i$. Though noisier, the randomness can sometimes help SGD escape local minima that trap standard gradient descent.

3 Gaussian Distribution

Dealing with randomness is a core component of data science. Data will inherently have a random element, so it's essential to differentiate between "signal" and "noise."

3.1 The Gaussian Probability Density Function

The Gaussian Probability Density Function (PDF) is defined as:

$$\phi_{\mu,\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where μ is the mean and σ^2 the variance. The Gaussian distribution is often abbreviated as $N(\mu, \sigma^2)$, commonly known as the "Normal" distribution.

This bell curve describes the relative likelihood of different numbers, and the area under the curve over any interval $[a, b]$ represents the probability of the number lying within that interval:

$$\int_a^b \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx.$$

The integral over the entire real line equals 1, ensuring that the distribution is normalized.

3.2 The Central Limit Theorem

Why is the Gaussian distribution so common? The Central Limit Theorem (CLT) explains this. Informally, it states that the sum of "many," "small," "independent" random variables will approximately follow a Gaussian distribution. This makes the Gaussian distribution especially useful in practice, as it simplifies mathematical treatment in many cases.

3.3 Moments of the Gaussian Distribution

The parameters μ and σ^2 are not just arbitrary; they have physical interpretations:

- The **mean** μ is the balancing point of the distribution, satisfying:

$$\mu = \int_{-\infty}^{\infty} x\phi_{\mu,\sigma^2}(x)dx.$$

- The **variance** σ^2 measures the spread of the distribution around the mean:

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2\phi_{\mu,\sigma^2}(x)dx.$$

3.4 Python Implementation

Random numbers from a Gaussian distribution can be generated in Python using the ‘numpy‘ library:

```
import numpy as np
x = np.random.normal(mu, sigma)
```

These random numbers are pseudo-random, generated based on a deterministic seed, making the process repeatable if the seed is set manually.

3.5 Realizations, Random Variables, and Distributions

Definition 3.1. A **realization** is a particular observed value of a random variable.
Definition 3.2. A **random variable** is a placeholder for an unknown random quantity before its value is revealed.
Definition 3.3. A **distribution** describes the relative likelihood of different realizations of a random variable.

3.6 Linear Models of a Single Random Variable

In many applications, output variables are modeled as affine transformations of input variables. For example, if $X \sim N(\mu, \sigma^2)$, and $Y = aX+b$, then $Y \sim N(a\mu+b, a^2\sigma^2)$.

3.7 The Expectation Operator

The expectation, or expected value, of a function $g(X)$, where $X \sim N(\mu, \sigma^2)$, is:

$$E[g(X)] = \int_{-\infty}^{\infty} g(x)\phi_{\mu,\sigma^2}(x)dx.$$

The expectation operator is linear, i.e.,

$$E[aX + b] = aE[X] + b.$$

4 Bivariate Gaussian Distribution

The bivariate Gaussian distribution describes a joint distribution of two random variables. The PDF for the bivariate Gaussian distribution with parameters μ_1, μ_2, σ_1^2, σ_2^2, and ρ (the correlation coefficient) is:

$$\phi_{\mu_1,\mu_2,\sigma_1^2,\sigma_2^2,\rho}(x_1,x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\Bigg(-\frac{1}{2(1-\rho^2)}\Bigg[\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2 - 2\rho\frac{(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \left(\frac{x_2-\mu_2}{\sigma_2}\right)^2$$

The contours of the bivariate Gaussian distribution are ellipses, and the parameter ρ controls the orientation and shape of these ellipses.

4.1 Linear Models with Bivariate Gaussians

A linear model of two Gaussian random variables, $Y = a_1X_1 + a_2X_2 + b$, results in $Y \sim N(a_1\mu_1 + a_2\mu_2 + b, a_1^2\sigma_1^2 + a_2^2\sigma_2^2 + 2\rho a_1a_2\sigma_1\sigma_2)$.

5 Statistics: Inference from Data

In practice, we often need to infer distribution parameters from observed data. The **maximum likelihood estimate** (MLE) is a popular method for estimating parameters from data. For instance, given observations from a normal distribution, the MLE for the mean μ is the sample mean, and for the variance σ^2, it is the sample variance.

5.1 Testing for Normality

To test whether data follows a Gaussian distribution, visual methods such as plotting histograms and comparing them with Gaussian PDFs, or using normal probability plots, are effective low-tech approaches.

6 Gaussian Distribution

6.1 Probability Intro

Want to to model uncertainy in outcomes + situations where outcomes are not known prior.

Discrete probability:- $\mathcal{X}$:= set of outcomes, where p is the probability of the outcome happening

- $p(x) \geq 0$ for all $x \in \mathcal{X}$
- $\sum_{x \in \mathcal{X}} p(x) = 1$ – the **Law of Total Probability**

Random variable X has outcomes $\mathcal{X} \sim p$ when $\mathcal{X}$ is the (apriori unknown) outcome drawn from p. An **event** is any subset of $\mathcal{X}$ (all possible outcomes), usually denoted by A or E.

The **expectation** $\mathbb{E}$ of a random variable X is its expected value. Note that

$$\mathbb{E}[x] = \sum_{x \in \mathcal{X}} \mathbb{P}(X = x) \times x = \sum_{x \in \mathcal{X}} p(x) \times x$$

Additionally,

- For a function $g(x)$, $\mathbb{E}[g(x)] = \sum_{x \in \mathcal{X}} p(x) \times g(x)$.
- **Linearity of Expectation** states that for any random variables X_1 and X_2, $\mathbb{E}[X_1 + X_2] = \mathbb{E}[X_1] + \mathbb{E}[X_2]$

Two random variables are **independent** if

$$\mathbb{P}(X_1 = x_1, X_2 = x_2) = \mathbb{P}(X_1 = x_1) \times \mathbb{P}(X_2 = x_2)$$

6.2 Discrete + Continuous Distributions

Discrete Distributions

$\mathcal{X}$ - a discrete set
p:-

- $p(x) \geq 0$ for all $x \in \mathbb{R}$
- $\sum_{x \in \mathcal{X}} p(x) d(x) = 1$

$$\mathbb{E}[X] = \sum_{x \in \mathcal{X}} x \times p(x)$$

For an event $A \subseteq \mathcal{X}$,

$$\mathbb{P}(X \in A) = \mathbb{P}(A) = \sum_{x \in A} p(x)$$

Continuous Distributions

$\mathcal{X} \sim$ a continuous set of real numbers
p:-

- $p(x) \geq 0$ for all $x \in \mathbb{R}$
- $\int_{-\infty}^{\infty} p(x) \times x = 1$

$$\mathbb{E}[x] = \int_{\mathbb{R}} p(x) \times x dx$$

For an event $A \subseteq \mathcal{X}$,

$$\mathbb{P}(X \in A) = \mathbb{P}(A) = \int_{x \in A} p(x) dx$$

6.3 Gaussian Distributions

This is the most important distribution, also known as the **bell curve** or **normal distribution**.

Definition 6.1. A **Gaussian distribution**with mean μ and variance $\sigma^2 > 0$ has the following distribution:

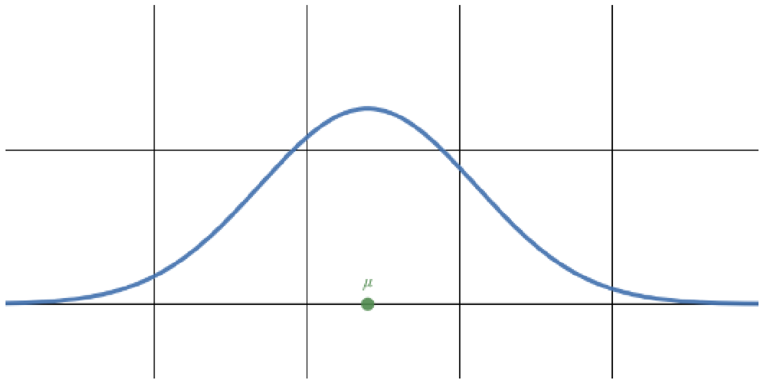
$$p(x) = \phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

From MATH2411 – The **normal distribution**, also called the **Gaussian distribution**, is the most important continuous probability distribution because of its ubiquity in the real world.

Definition 6.2 (Normal Distribution)**.** A **normal distribution** has notation: $X \sim N(\mu, \sigma^2)$ where $\mu \in (-\infty, \infty)$ is the *mean* and $\sigma^2 \in (0, \infty)$ is the **variance**. A normal distribution has pdf:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \text{ for all real values x}$$

A normal distribution has expectation $E(X) = \mu$ and variance $Var(X) = \sigma^2$. Note that all normal distributions have a bell-shaped density curve regardless of the values of μ and σ.



Observe that $\phi_{\mu,\sigma^2}(x)$ is symmetric about μ and that $X \sim \phi_{\mu,\sigma^2} \to \text{N}(\mu, \sigma^2)$

Definition 6.3. The **central limit theorem (CLT)** states that a random quantity that is the sum of many small independent random quantities will have follow a Gaussian distribution.

By definition of **variance**: $\mathbb{E}[(x-\mu)^2]$, a normal distribution has

$$\mathbb{E}[x] = \mu, \text{Var}(x) = \sigma^2$$

Note that with expectation and variance, given an $X \sim p(x)$, $Y = aX + b$,

$$\mathbb{E}[Y] = a\mathbb{E}[X] + b, \text{Var}(Y) = a^2\text{Var}(X)$$

6.4 Linear Models

A large part of data science is reasoning about input/output relationships in a system. We're going to look at input which follows a Gaussian distribution.

Theorem 6.1. *With $X \sim N(\mu, \sigma^2)$ and $Y = aX + b$, then*

$$Y \sim N(a\mu + b, a^2\sigma^2)$$

Note that we can standardize Gaussian distributions: From MATH2411-

Definition 6.4 (Standard Normal Distribution)**.** The **standard normal distribution** has mean 0 and variance 1, i.e., $N(0,1)$. The random variable following the standard normal distribution is often denoted by Z in probability and statistics. Its probability density function (pdf) is

$$\varphi(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

and its distribution function (cdf) is

$$\Phi(z) = \int_{-\infty}^{z} f(t)dt = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt = \mathbb{P}(X \leq z)$$

Note that $\lim_{x\to\infty} \Phi_{\mu,\sigma^2}(x) = 1$ and recall that Φ_{μ,σ^2} is an increasing function (cdf!).

To **standardize** a distribution, we use

$$\frac{x-\mu}{\sigma} \sim N(0,1)$$

6.5 Random Vectors

Tool to analyze how different random quantities are related (e.g., current and voltage, image darkness and malignancy, etc).

Definition 6.5. A **random vector** is a placeholder for an unrevealed vector-valued random quantity: either thought of as a *single* random entity or a *vector* of random variables.

For any p over $\mathbb{R} \times \mathbb{R}$,

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x_1, x_2)dx_1dx_2 = 1$$

6.5.1 Bivariate Gaussian

Definition 6.6. The bivariate $\mathcal{N}$ distribution with paraneters $\mu_1 \in \mathbb{R}, \mu_2 \in \mathbb{R}, \sigma_1^2 > 0, \sigma_2^2 > 0, -1 < \rho < 1$ is

$$\phi_{\mu_1,\mu_2,\sigma_1^2,\sigma_2^2,\rho}(\overrightarrow{x}) = \frac{\exp[-\frac{1}{2(1-\rho^2)}((\frac{x_1-\mu_1}{\sigma_1})^2 - 2\rho(\frac{x_1-\mu_1}{\sigma_1})(\frac{x_2-\mu_2}{\sigma_2}) + (\frac{x_2-\mu_2}{\sigma_2})^2)]}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}}$$

ρ is the parameter that characterizes how X_1 and X_2 relate to each other. If

- $\rho = 0$, then X_1 and X_2 are independent (Gaussian)
- $\rho > 0$, then X_1 and X_2 are *positively* correlated
- $\rho < 0$, then X_1 and X_2 are *negatively* correlated

If $A \subseteq \mathbb{R} \times \mathbb{R}$, then

$$\mathbb{P}((x_1, x_2) \in A) = \iint_{(x_2, x_2) \in A} \phi_{\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho}(x_1, x_2) dx_1 dx_2$$

6.5.2 Marginalization

X_1 over $\mathcal{X}_1$, X_2 over $\mathcal{X}_2$, where $p(x_1, x_2)$ = joint distirbution of X_1, X_2.

7 Data Acquisition and Cleansing

7.1 Introduction

Data acquisition and cleansing is the first stage of the data science workflow. While foundational in its importance, it is also often the most time-consuming phase. A significant portion of a data scientist's work involves preparing the data to be analyzed, and this step must be done carefully to avoid introducing errors. Ironically, despite its critical nature, we will not spend much time on this stage as there is no universal method for data cleansing. Each dataset has its unique challenges, and practical experience is often the best guide.

7.2 File I/O in Python

To analyze data, the first step is typically loading it into Python. The most straightforward way to do this is by reading from files using Python's `open()` function. For example, to read a text file:

```
f = open('sometext.txt', 'r')
```

The 'r' flag indicates that the file is being opened in read mode. Python also provides several ways to read the file contents:

- `read(n)` reads the next *n* bytes or characters.
- `readline()` reads a single line of the file.
- `readlines()` reads the entire file and returns a list of lines.

Once you have finished reading from a file, it is important to close it using:

```
f.close()
```

For binary data, such as images or executables, use the 'b' flag:

```
f = open('file.dat', 'rb')
```

Python handles the details of how different operating systems represent newline characters (e.g., Windows uses `\r\n`, Unix uses `\n`). However, when working with binary data, it is necessary to handle these details manually, especially when writing portable code.

7.3 Unicode

In a world where datasets often include multiple languages and special symbols, handling text encoding is crucial. Text is commonly encoded in **Unicode**, which provides a way to represent thousands of characters from different languages and scripts.

The most common encoding formats include:

- **UTF-8**: A variable-length encoding that uses 1 to 4 bytes per character. It is backward compatible with ASCII, making it the most common format for web data.
- **UTF-16**: Uses 2 or 4 bytes per character, balancing memory usage and readability.
- **UTF-32**: A fixed-width encoding using 4 bytes per character, which is simple to process but memory-inefficient.

Python 3 has built-in support for Unicode, and you can specify the encoding when opening a file:

```
f = open('unicodefile.txt', encoding='utf-8')
```

This prevents encoding errors and ensures that text data is properly interpreted. However, determining the correct encoding for an unknown file can involve trial and error or the use of tools like `hexdump`.

7.4 Reading CSV Files

CSV (Comma-Separated Values) files are widely used for storing tabular data. Each line of a CSV file represents a row in a table, with columns separated by commas. Despite its simplicity, handling CSV files can become tricky when cells contain commas or newlines. Python's `csv` module helps manage such files, allowing for custom delimiters and quoting styles.

To read a CSV file:

```
import csv
with open('data.csv', 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

CSV reading can be customized by passing dialect options, which allows the module to handle variations in how different software generates CSVs (e.g., using tabs or semicolons instead of commas).

7.5 Sampling Bias

Sampling bias occurs when the data collected does not represent the entire population. One famous example is the 1948 U.S. election, where polls erroneously predicted Dewey's victory over Truman due to bias introduced by calling people who owned telephones—a luxury at the time associated with wealthier voters who tended to support Dewey.

In modern times, sampling bias can also stem from differences in education level or other factors. For example, in the 2016 U.S. election, more educated individuals were more likely to participate in polls, which skewed results toward Clinton.

Sampling bias can also occur in time-based processes. Consider trying to measure the average time between events in a random process. If you simply measure the time to the next event and double it, the estimate will be biased toward longer intervals. Correcting for such biases requires techniques like **Palm theory**or weighting responses based on the sample's deviation from the population.

7.6 Untruthful Respondents

When polling sensitive topics, respondents may lie, especially if the truthful answer is socially undesirable. To address this, the **randomized response**method offers a clever workaround. Respondents flip a coin and answer "yes" regardless of the truth if it shows heads. If tails, they provide their true answer. This way, individuals maintain anonymity while surveyors can infer the true distribution of responses through statistical analysis.

7.7 Aliasing

Aliasing arises when a signal is sampled too slowly, leading to an incorrect interpretation of its frequency. This phenomenon is famously seen in old movies where wagon wheels appear to rotate backward. This happens when the frame rate is lower than the speed of the wheel's rotation, causing points on the wheel to seem to move in the opposite direction.

The solution to aliasing is to ensure that the sampling rate exceeds the Nyquist rate, which is twice the maximum frequency of the signal. This ensures that enough data points are captured to reconstruct the original signal accurately.

7.8 Outliers

Outliers are data points that fall far outside the expected range based on a given distribution. Detecting and handling outliers is a crucial part of data cleansing, as they can significantly skew results if left unchecked.

One method for identifying outliers is **Chauvenet's Criterion**, which works under the assumption of a normal distribution. It calculates how many standard deviations (*z-scores*) a data point is from the mean:

$$D_{\text{max}} = \max_i \frac{|x_i - \mu|}{\sigma}$$

The probability of a data point being this far from the mean can be calculated using the normal distribution's cumulative distribution function:

$$P(\text{outlier}) = 2\left[1 - \Phi_0, 1(D_{\text{max}})\right]$$

If the expected number of such points is less than 1/2, the point is considered an outlier. Once detected, outliers should be carefully investigated. While some may be due to data corruption, others might be valid and could suggest issues with the underlying assumptions of the model, such as the assumed distribution.

8 Data Exploration and Modeling: Regression

9 Prediction and Decision Making: Hypothesis Testing

10 Prediction and Decision Making: Classification

11 Time Series and DFT

12 Ethical Hazards in Data Science