# Chapter 1: The Role of Statistics in Biological Sciences

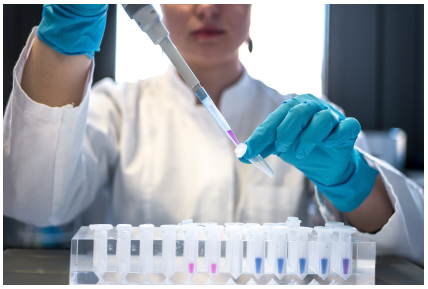
Learning goals

1. Understand why variability in biological data makes statistics essential for biological research.
2. Recognize how statistics helps in hypothesis testing and making inferences from sample data.
3. Understand the role of statistics in understanding correlations and making predictions in biological studies.
4. Appreciate the importance of learning and applying statistics in biological and scientific research.

## Introduction

In the vast, diverse world of biology, a primary goal is to seek answers to questions about life - its processes, structures, functions, and interactions. Biologists employ various methods, tools, and techniques to generate **data**, a collection of factual information, statistics, and/or observations, and draw conclusions from that data. However, due to the inherent complexity and variability of life, it's often challenging to interpret the biological data and make sense of it all. That's where statistics comes in.

**Statistics**, a branch of mathematics dealing with data collection, analysis, interpretation, presentation, and organization, provides crucial tools for biologists. By incorporating statistical analysis into their research, scientists can decipher patterns, assess reliability, and make predictions about their observations, enhancing their understanding of biological phenomena.




### Video


Video link

## Importance of Statistics in Biology

### Overcoming Variability

Life is complex and dynamic, full of variability at every turn. Whether studying genes, populations, or ecosystems, biologists encounter considerable natural variation. For instance, not all fruit flies have the same life span, and not all trees of a species grow to the same height. Statistics allow scientists to account for this variability and distinguish real effects from random fluctuations.

### Inference and Hypothesis Testing

Biologists often aim to draw general conclusions from studying a small subset or sample of a larger population. However, due to the inherent variability in biology, it's often challenging to determine whether an observed effect is a genuine phenomenon or simply occurred by chance. Statistics provides tools like hypothesis testing and confidence intervals, allowing scientists to infer about populations based on sample data, and quantify the degree of uncertainty in these inferences.

### Understanding Correlations and Predicting Outcomes

Often, biologists wish to understand relationships between **variables**, factors or elements that can be measured, manipulated, or controlled in order to understand its effect on a particular phenomenon or outcome. For example, how does rainfall affect plant growth? Or how does a mutation, a random change in an organisms genetic code, influence progression of a disease? Statistics offers tools like correlation and regression analysis to help quantify relationships between variables and predict future outcomes.

## A Biological Example

Let's consider an example from ecology, the study of interactions in nature: studying the impact of pollution on a fish population in a lake. Suppose an ecologist observes a decline in the fish population after a factory starts dumping waste into the lake. Is the decline due to pollution, or could it just be a natural fluctuation?

To investigate, the ecologist collects data from several polluted and unpolluted lakes over time. Using statistical techniques like hypothesis testing (Chapter 07), the ecologist can compare the fish populations in polluted vs. unpolluted lakes to determine if there's a significant difference. Furthermore, regression analysis

(Chapter 13) could be used to model the relationship between pollution levels and fish population size, predicting the fish population's response to varying pollution levels.

Such statistical analyses allow the ecologist to make objective, evidence-based claims about the pollution's effects, providing a basis for environmental policy decisions or remediation efforts.

## Conclusion

In the dynamic world of biology, where variability is a rule rather than an exception, statistics acts as a beacon, guiding scientists through a sea of data. It offers the tools to test hypotheses, infer population characteristics, explore relationships, and make predictions, thereby enabling meaningful interpretation of biological data.

As such, learning and understanding statistics is essential for anyone embarking on a scientific career. It's not just about crunching numbers; it's about telling the story hidden within the data, making sense of the complex world we inhabit, and contributing to scientific progress. As biologists and scientists, we have a responsibility to produce robust, reliable, and accurate interpretations of our observations. This goal can only be achieved through the diligent application of statistical reasoning and methods.

## End of chapter questions

1. Biologists encounter significant natural variation in their studies. How does incorporating statistical analysis help researchers overcome the challenge of variability in biological data?

2. In the context of biology, why is it important to understand the distinction between a population and a sample? How does statistical inference enable biologists to make conclusions about populations based on sample data? Discuss the role of hypothesis testing and confidence intervals in this process.

3. The chapter mentions the use of correlation and regression analysis to understand relationships between variables in biology. How can these statistical tools aid biologists in predicting outcomes and making informed decisions in their research? Think of a real-life biological scenario to illustrate your explanation that is different than one provided in the text.

4. Why is hypothesis testing crucial in determining whether a decline in the fish population is due to pollution or natural fluctuation? Discuss how regression analysis can be used to model the relationship between pollution levels and fish population size in this context.

5. The conclusion emphasizes the importance of statistical reasoning and methods in producing robust and reliable interpretations of biological data. In your opinion, why is it necessary for biologists and scientists to have a solid understanding of statistics? Discuss the potential implications of inaccurate or flawed statistical analysis in biological research.

# Chapter 2: Fundamentals of Data in Scientific Research: Variables, Collection, and Bias

Learning goals

1. Understand and identify the different types of variables.
2. Understand the different methods of data collection and their respective benefits and limitations.
3. Distinguish between population parameters and sample estimates.
4. Understand the implications of sampling error and bias on the quality of data and how to mitigate them.

## Introduction

In scientific research, the power of inquiry is inextricably linked with the data we gather. Data, or information collected for reference or analysis, is the cornerstone that enables scientists to generate new knowledge and insight. This chapter will guide you through the essentials of data, the different types of variables, how data is collected, and the key considerations when dealing with population and sample data.

### Types of Variables

Understanding the types of data, or variables, is the first step in effective data analysis. Variables can broadly be classified as numerical and categorical.

1. **Numerical variables** represent measurements or counts and can be subdivided into two types:

- **Discrete:** These variables take on only certain values within a given range, such as the number of petals on a flower or the number of cells in a tissue sample.
- **Continuous:** These variables can take on any value within a range, like the weight of an organism or the pH level of a solution.

2. **Categorical variables** represent qualitative data and identify a particular group or category to which data points belong. Examples include the species of a plant, the type of a disease, or the color of a bird's feathers.

## Data Collection Methods

Collecting accurate and relevant data is critical for scientific investigations. The two primary study designs for data collection are experimental and observational studies:

1. **Experimental studies** involve designing an experiment with specific treatments and controls for outside variables. This design is useful for determining cause and effect relationships but may be challenging to generalize to broader, real-world contexts.

2. **Observational studies** involve making observations in the real world without intervening. These studies are good for identifying patterns but don't control for outside variables, making it difficult to infer causality.

## Video


Video Link

## Population vs. Sample

Data can be collected from a complete group, known as the population, or a subset of that group, called a sample.

1. **Population (N):** This is the complete set of data from a particular source. Any conclusions drawn directly from a population are called parameters. These are constant and exact.
2. **Sample (n):** This is a subset of data drawn from the population for analysis. Conclusions from sample data, termed estimates, are random and approximate.

## Video


Video link

## Sampling Errors and Bias

When dealing with samples, two key considerations are sampling error and bias:

1. **Sampling error** refers to the random discrepancies between the sample estimate and the true population parameter. The larger the sample size, the smaller the sampling error, as the sample becomes more representative of the population.
2. **Sampling bias** occurs when certain members of the population are more likely to be sampled than others. This bias can be mitigated by ensuring samples are selected randomly. However, achieving true randomness can be challenging in certain scenarios.

Several common sources of sampling bias include:

- **Sample of convenience:** Researchers select the easiest samples to collect.
- **Volunteer bias:** Data comes only from members of the population who voluntarily provide data, such as respondents to a survey.

## Video


Video link

## Conclusion

Understanding data and its fundamentals is crucial for any scientific endeavor. This chapter has explored the types of variables, methods of data collection, and considerations for dealing with population and sample data. As you move forward in your exploration of scientific research, keep these principles in mind to ensure your findings are robust, reliable, and insightful.

## End of chapter questions

1. Why is it important for scientists to understand the different types of variables in data analysis? Discuss how the classification of variables as numerical or categorical influences the choice of appropriate statistical methods and the interpretation of results. Provide examples from scientific research to support your answer.

2. Compare and contrast experimental and observational study designs, highlighting their respective strengths and limitations. In what situations would each design be most suitable for gathering scientific data?

3. Explain the distinction between a population and a sample in data collection. Discuss why researchers often work with samples rather than attempting to collect data from entire populations. What are the advantages and challenges associated with using sample data to make inferences about populations?

4. Sampling error and bias are important considerations when working with sample data. Define sampling error and explain how it impacts the accuracy of sample estimates in relation to the true population parameters. Discuss strategies that researchers can employ to reduce sampling error.

5. The chapter mentions various sources of sampling bias, such as convenience sampling and volunteer bias. Choose one of these biases and discuss its potential impact on the validity and generalizability of research findings. Suggest alternative sampling methods that researchers can employ to minimize the effects of this bias.

# Chapter 03: Displaying data

Learning goals

- Understand the importance of data visualization in biostatistics.
- Create and interpret histograms, scatter plots, bar plots, violin plots, and time series plots using Python.
- Apply best practices for data visualization to create clear, effective plots.

## Importance of Data Visualization

Data visualization is a key component of any data analysis. It allows us to see patterns and trends in the data that might not be immediately apparent from the raw numbers. For biostatisticians, this can be particularly important when dealing with large, complex datasets. By visualizing the data, we can gain insights that can guide our analysis and help us make informed decisions.

## Creating Plots in Python

Python, with libraries such as `Matplotlib` and `Seaborn`, provides powerful tools for creating a wide range of data visualizations. In this chapter, we have explored how to create several types of plots, including histograms, scatter plots, bar plots, violin plots, and time series plots. Each of these can be useful for visualizing different types of data and answering different types of questions.

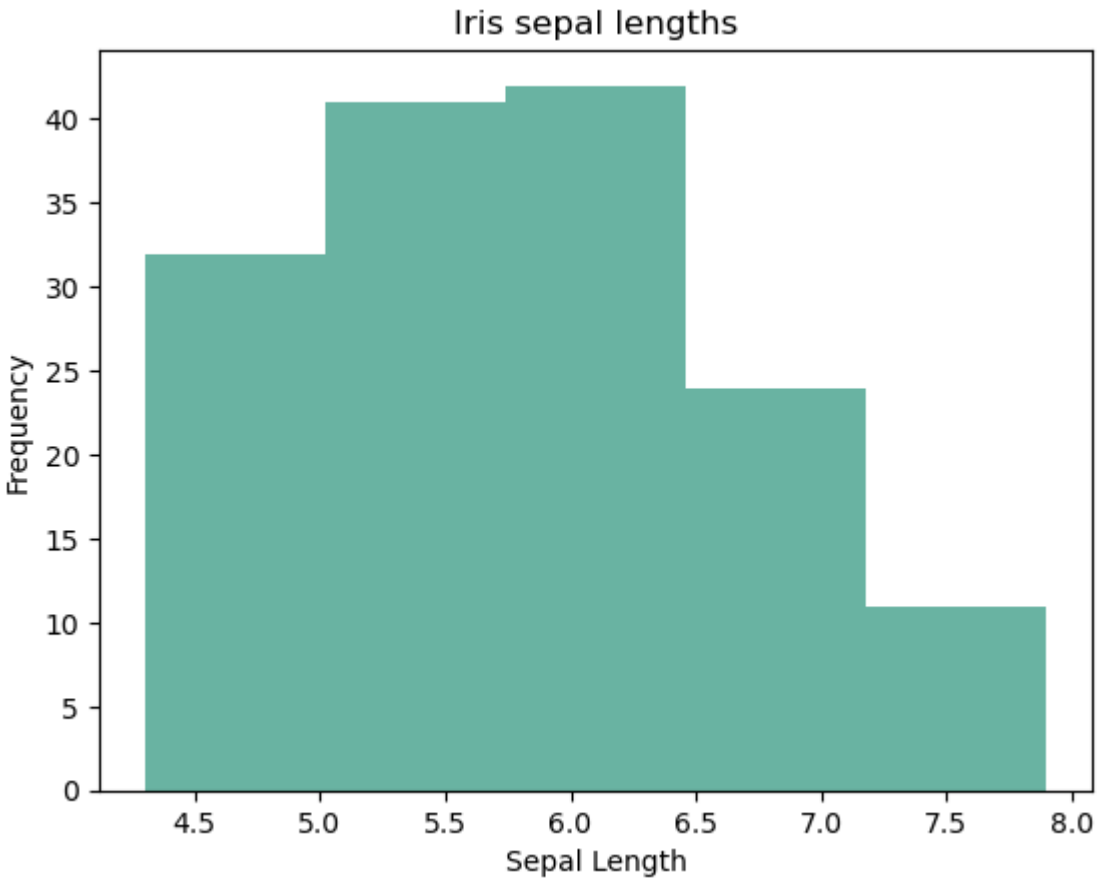
### Histogram

A histogram is a graphical representation that organizes a group of data points into a specified range. It is an accurate method to represent the distribution of numeric data. It's best for visualizing the distribution of continuous data. Stylistically, it's important to choose an appropriate number of bins because too few can oversimplify the data, while too many can overcomplicate the data.

Below is a histogram of the lengths of sepals of iris flowers. The `iris` dataset is imported from the `seaborn` Python library.

In [9]:
```
# Histograms in Python
# Call the data
import seaborn as sns
iris = sns.load_dataset('iris')
```

```
# Built-in plot() function
x = iris['sepal_length']

# Using matplotlib for histogram
import matplotlib.pyplot as plt
plt.hist(x, color = "#69b3a2", bins = 5) # You can change the "bins" value smooth out the patter
plt.xlabel('Sepal Length')
plt.ylabel('Frequency')
plt.title('Iris sepal lengths')
plt.show()
```



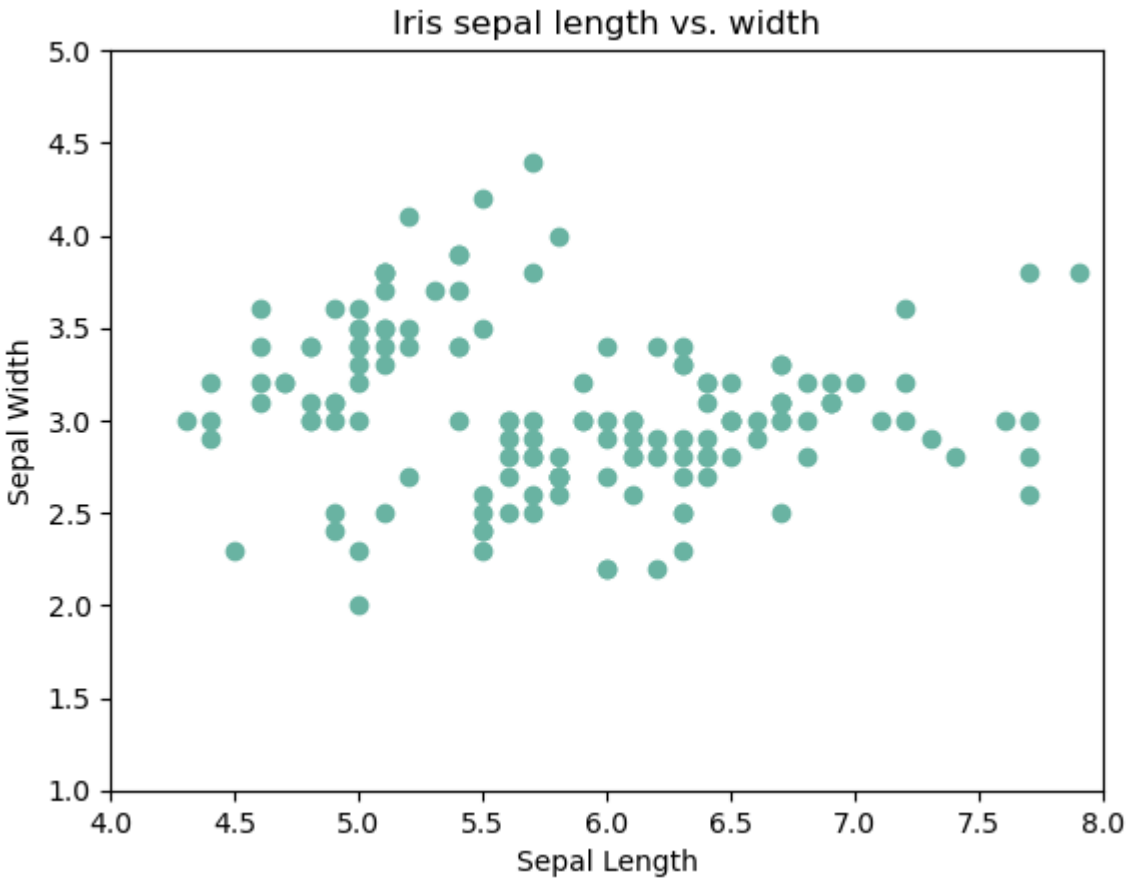

## Scatter Plot

A scatter plot uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables. They are best for visualizing the relationship between two continuous variables. Stylistically, it's important to label each axis and provide a title for the plot.

Below is an example scatter plot comparing *Iris* flower sepal length and sepal width from the `iris` dataset.

In [10]:
```
# Scatter plots in Python
import seaborn as sns
iris = sns.load_dataset('iris')

# Call the data
```

```
x = iris['sepal_length']
y = iris['sepal_width']

# Python will automatically detect the data type and produce
# a scatter plot
plt.scatter(x, y, color = "#69b3a2")
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Iris sepal length vs. width')
plt.xlim(4, 8)
plt.ylim(1, 5)
plt.show()
```

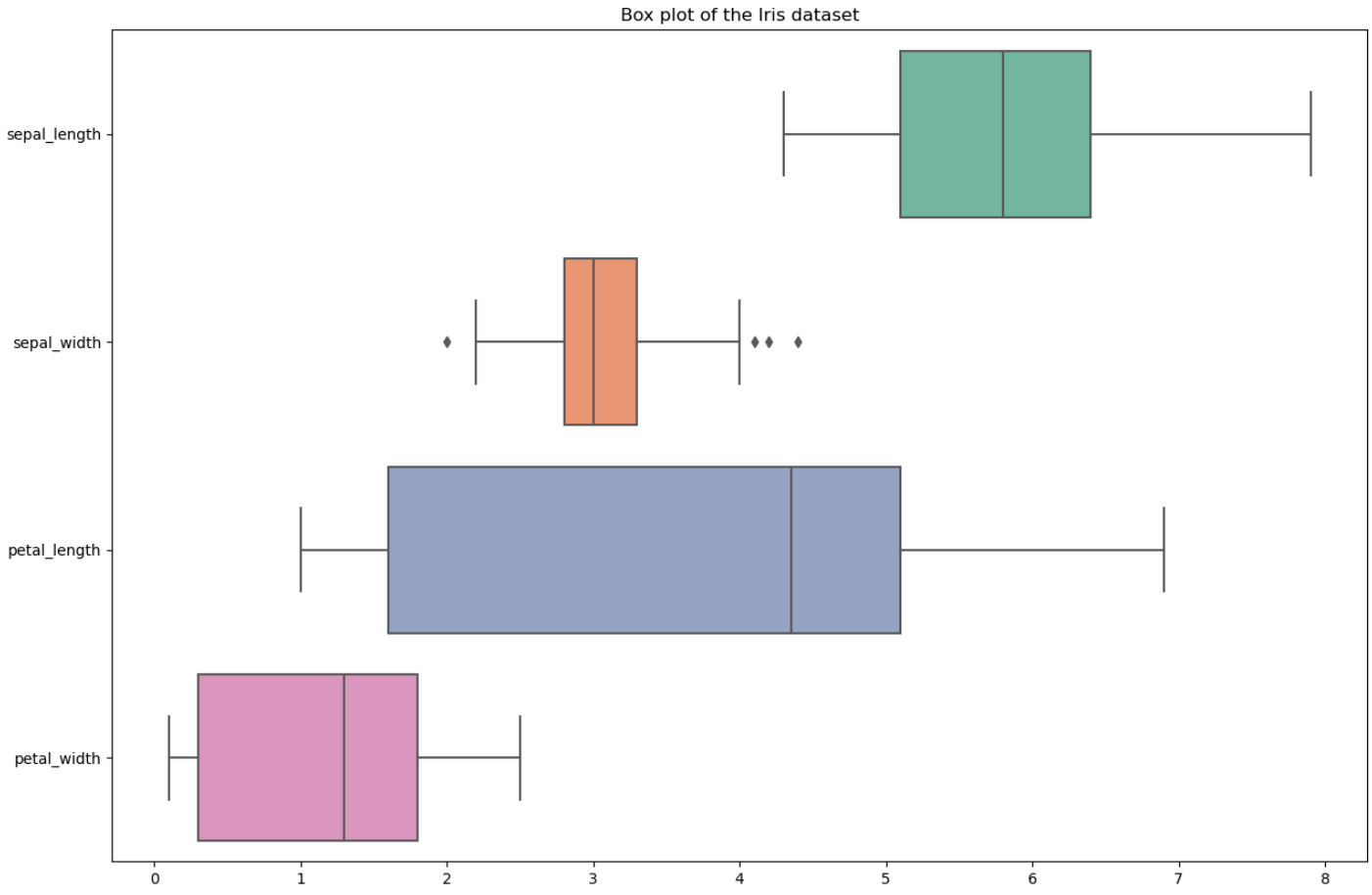



## Box and whisker plots

In a box and whisker (or just box) plots, the box represents the interquartile range (IQR), which is the range between the first quartile (25th percentile) and the third quartile (75th percentile). The line inside the box represents the median (50th percentile). The whiskers represent the range of the data within 1.5 times the IQR. Any data points outside this range are considered outliers and are represented as individual points. Box plots are great for visualizing the distribution and skewness of your data, as well as the presence of outliers. They can be used for both univariate analysis (looking at one variable at a time) and multivariate analysis (looking at two or more variables together). Make sure to label the x and y axes so it's clear what each box plot represents and include a title.

Below we take a look at the spread of the four *Iris* flower measurements found in the `iris` dataset.

In [34]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Load the iris dataset
iris = sns.load_dataset('iris')

# Create a box plot for each feature in the iris dataset by species
plt.figure(figsize=(15,10))
sns.boxplot(data=iris, orient="h", palette="Set2")
plt.title('Box plot of the Iris dataset')
plt.show()
```



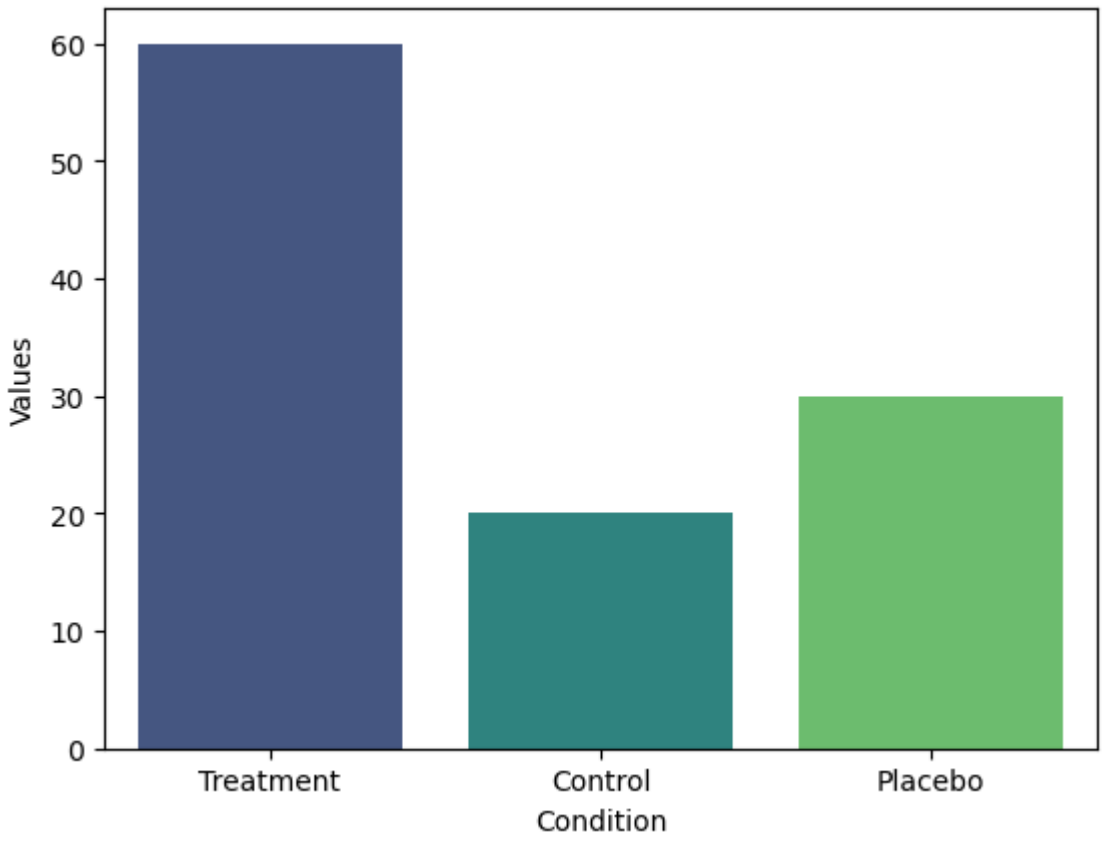

# Bar Plot:

A bar plot represents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. They are best for comparing the quantity, frequency, or other measure (like the mean) for different categories or groups. Stylistically, it's important to label each bar or group, and the plot should include a title and labels for each axis.

The below bar plots represent observations of some value from three experimental conditions.

In [14]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

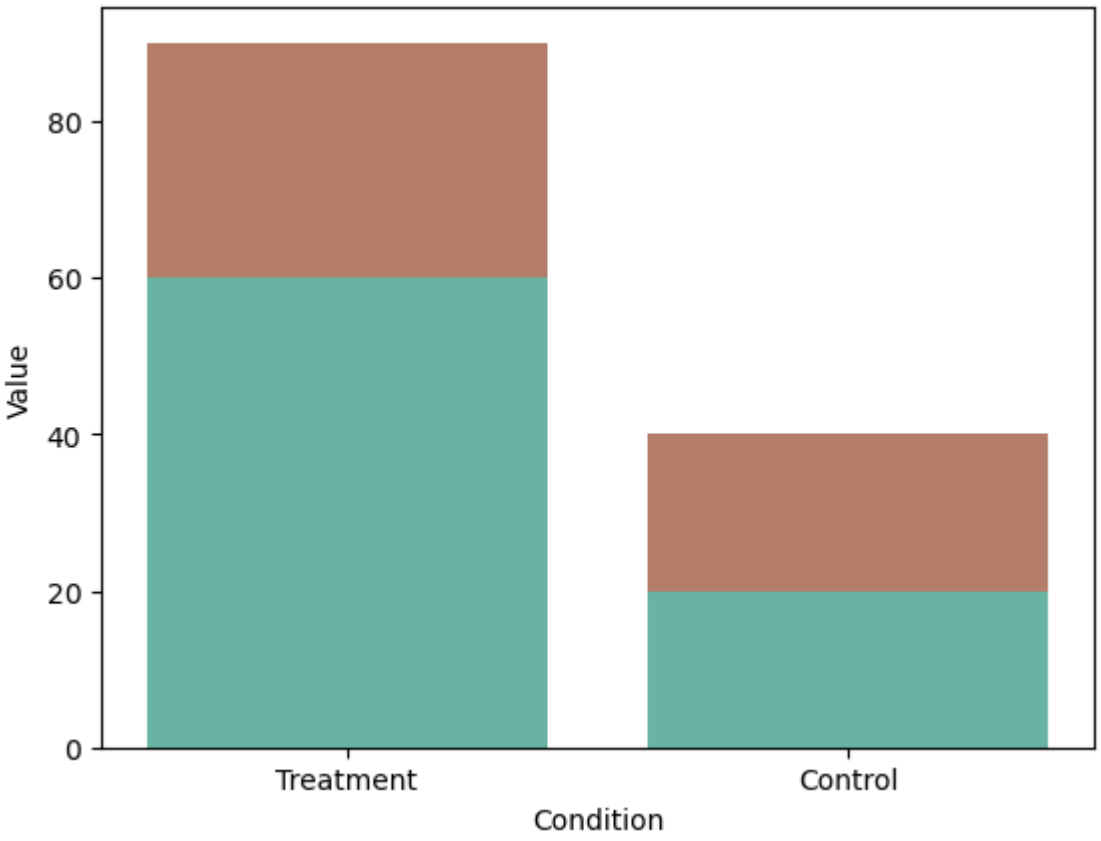
```
# Sample data for a single group
values = [60, 20, 30, 20]
condition = ["Treatment", "Control", "Placebo", "Control"]

data = pd.DataFrame({'Values': values, 'Condition': condition})

# Create the figure and axes manually
fig, ax = plt.subplots()

# Single group bar graph
sns.barplot(x='Condition', y='Values', data=data, ax=ax, palette='viridis')

# Show the plot
plt.show()
```




**Stacked bar plot** for multiple groups

In [13]:
```
import matplotlib.pyplot as plt
import seaborn as sns

values = [60, 20, 30, 20]
condition = ["Treatment", "Control", "Treatment", "Control"]
group = ["Group 1", "Group 1", "Group 2", "Group 2"]

data = pd.DataFrame({'Values': values, 'Condition': condition, 'Group': group})

# Create a stacked bar plot
group1 = data[data['Group'] == 'Group 1']['Values']
group2 = data[data['Group'] == 'Group 2']['Values']
```

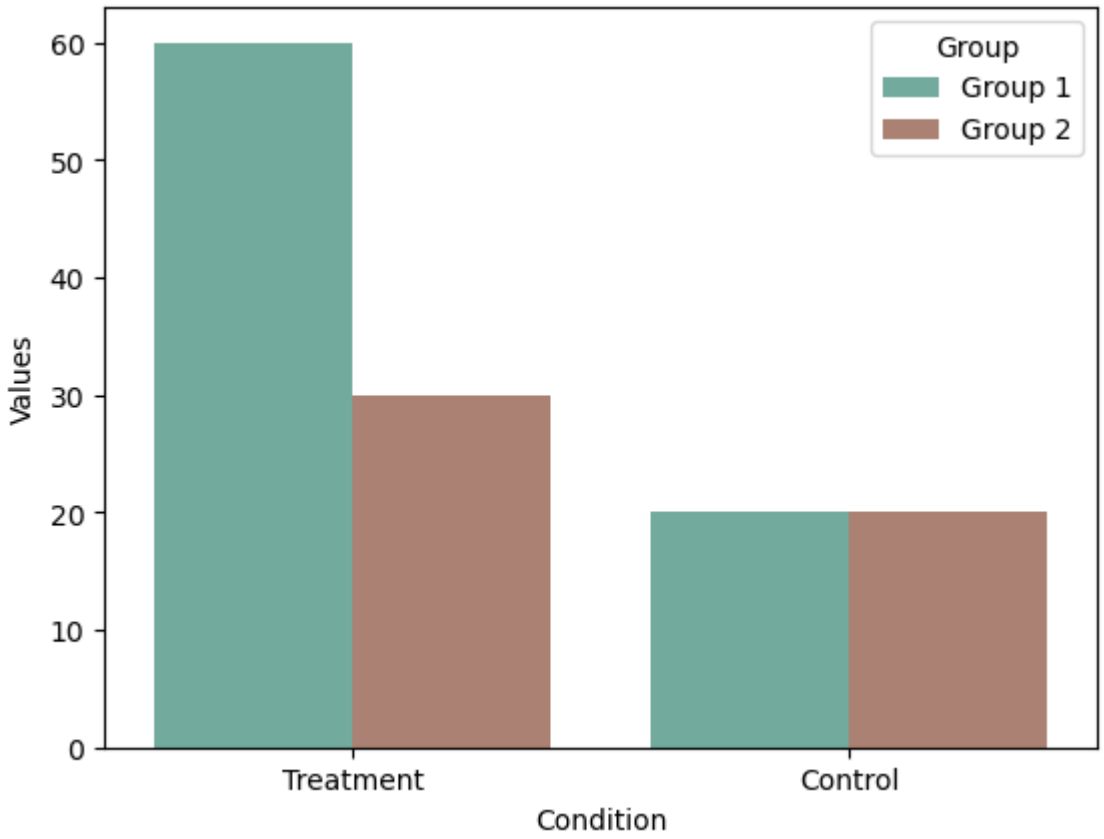
```
# Create a bar plot for 'Group 1'
plt.bar(data['Condition'].unique(), group1, color = "#69b3a2")
# Stack 'Group 2' on top of 'Group 1'
plt.bar(data['Condition'].unique(), group2, bottom=group1, color = "#b37d69")
plt.xlabel('Condition')
plt.ylabel('Value')
plt.show()
```




**Side by side bar plot** for multiple groups

In [11]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

values = [60, 20, 30, 20]
condition = ["Treatment", "Control", "Treatment", "Control"]
group = ["Group 1", "Group 1", "Group 2", "Group 2"]

data = pd.DataFrame({'Values': values, 'Condition': condition, 'Group': group})

# Create the figure and axes manually
fig, ax = plt.subplots()

# Bars side by side
sns.barplot(x='Condition', y='Values', hue='Group', data=data, ax=ax, palette=["#69b3a2", "#b37d
```

```
# Show the plot
plt.show()
```

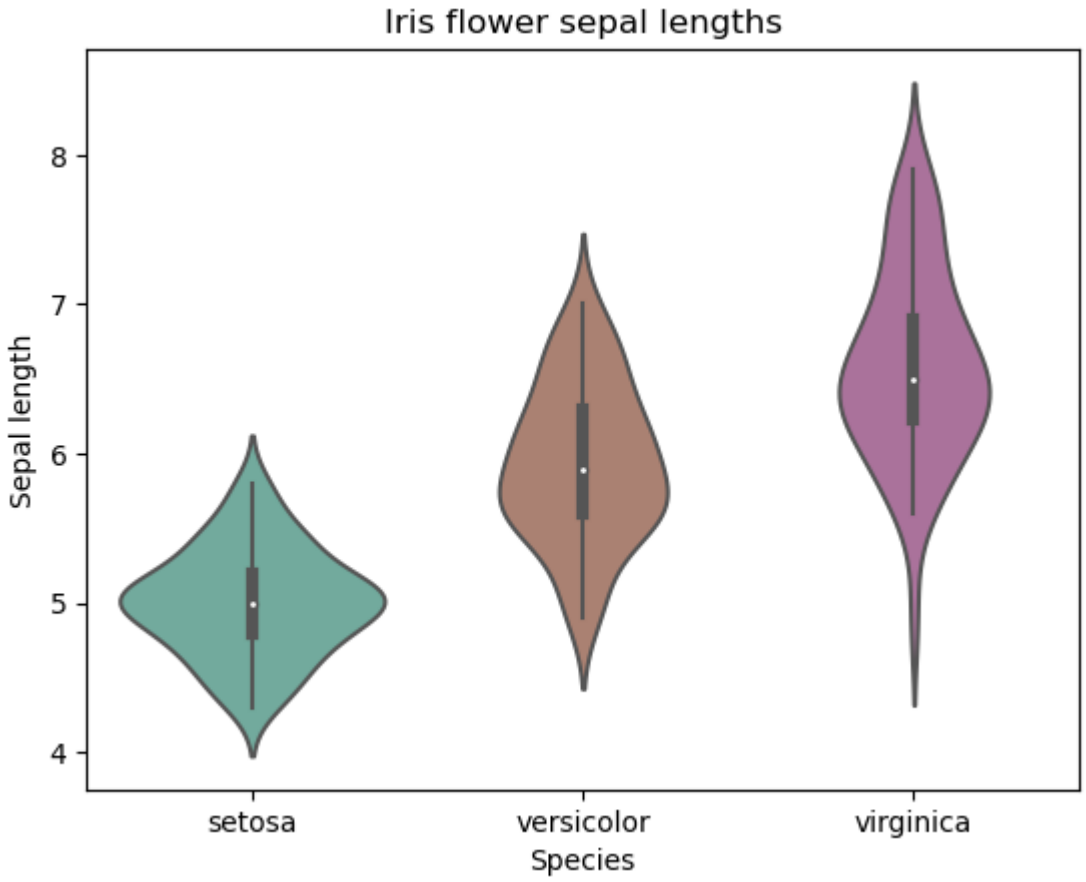



## Violin Plot

A violin plot is a method of plotting numeric data combining the box plot and kernel density plot. It's used to visualize the distribution of numerical data of different variables. It is best for comparing the distribution of data between groups. Stylistically, it's important to label each group and axis, and the plot should include a title.

Below we see the distribution of observations of sepal lengths for three different species of *Iris* flowers.

In [22]:
```
# Violin plots using seaborn library
import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset('iris')

sns.violinplot(x = 'species', y = 'sepal_length', data = iris, palette = ["#69b3a2", "#b37d69",
plt.xlabel('Species')
plt.ylabel('Sepal length')
plt.title('Iris flower sepal lengths')
plt.show()
```

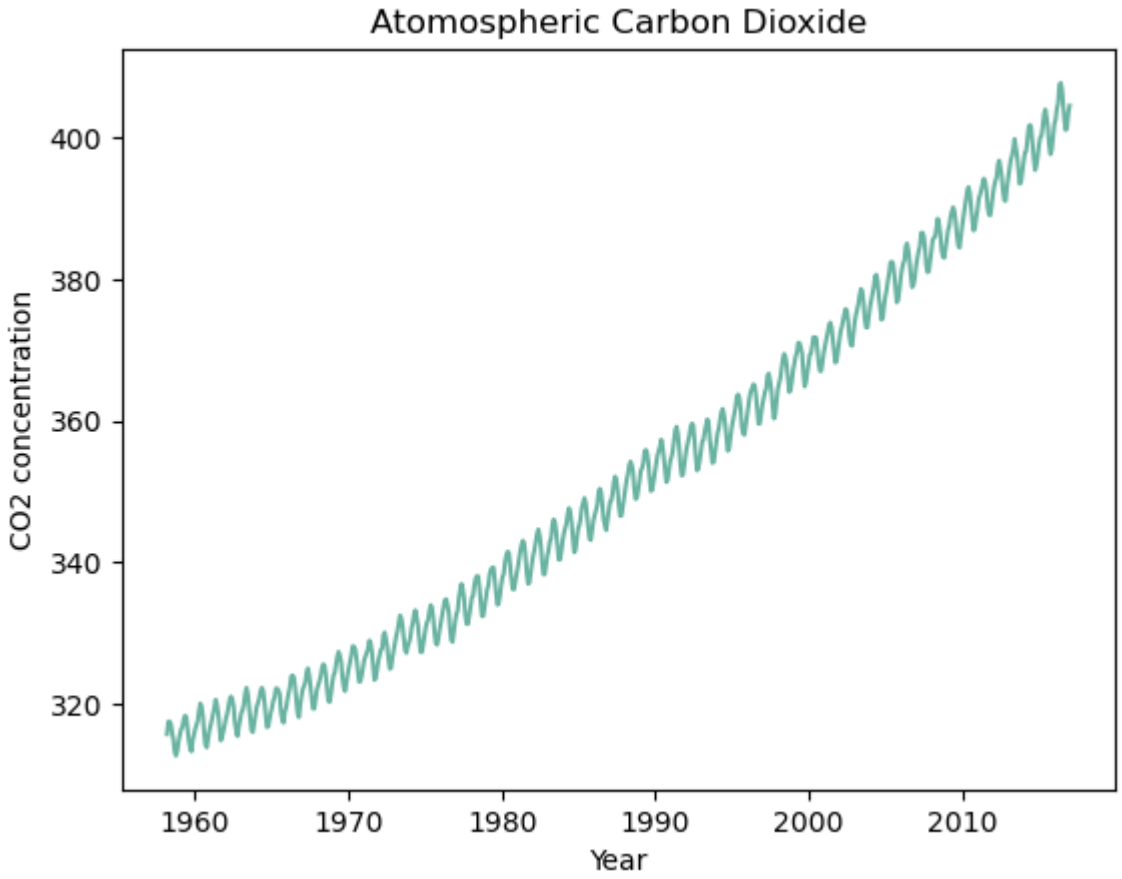



## Time Series Plot

A time series plot is a graph where some measure of time is the unit on the x-axis, often called the time-axis. The y-axis is for the variable that is being measured over time. It's best for visualizing how a variable changes over time. Stylistically, it's important to clearly label the time points and ensure that the time scale is appropriate for the data.

Here we see the change in atmospheric carbon dioxide over time.

In [2]:
```
# Time series in Python
import pandas as pd
import matplotlib.pyplot as plt

co2 = pd.read_csv('https://raw.githubusercontent.com/datasets/co2-ppm/master/data/co2-mm-mlo.csv

plt.plot(co2['Decimal Date'], co2['Interpolated'], color = "#69b3a2")
plt.xlabel('Year')
plt.ylabel('CO2 concentration')
plt.title('Atomospheric Carbon Dioxide')
plt.show()
```




## Creating Tables in Python

Most data is stored in some form of table format. These formats include .csv (comma separated values), .xlsx (Microsoft Excel), and .tsv (tab separated values). You can import tables from local storage or the internet using Python. You can also export data as a table that can then be shared with other researchers. The code below imports data from a research project where a student looked at the distance of parks from subway stations in New York City.

In [15]:
```
# Producing tables using pandas library
import pandas as pd

# Import data from the internet
subway_data = pd.read_csv('https://raw.githubusercontent.com/lundquist-ecology-lab/subway_map_ana

# Remove "id" and "units" column
subway_data = subway_data.drop(['id', 'units'], axis=1)

# Convert m2 to k2 and round
subway_data['b_area'] = round(subway_data['b_area'] / 1000000, 2)
subway_data['g_area'] = round(subway_data['g_area'] / 1000000, 2)

# Produce new table with more readable heading names
subway_data.columns = ["Borough", "Subway Stations", "Stations near green spaces", "Total borough
```

```
# Export table as .csv to work with in other programs or share
subway_data.to_csv("outputs/subway_table.csv", index = False)
```

**Display table in Jupyter Notebook using Markdown**

Tables can be displayed on websites and in Jupyter Notebooks by using the Markdown language. The Python library `IPython` can also be used to convert a data frame into a Markdown table.

In [33]:
```
from IPython.display import display, Markdown

# Assuming 'data' is your DataFrame
markdown_table = subway_data.to_markdown()

# To display the markdown table in the Jupyter notebook
display(Markdown(markdown_table))
```

| | Borough | Subway Stations | Stations near green spaces | Total borough area ($km^2$) | Total green area ($km^2$) |
|---|---|---|---|---|---|
| 0 | Manhattan | 152 | 64 | 58.6 | 10.36 |
| 1 | Brooklyn | 167 | 38 | 180.24 | 15.43 |
| 2 | Queens | 80 | 18 | 282.3 | 21.5 |
| 3 | Bronx | 71 | 20 | 108.86 | 22.69 |
| 4 | Staten Island | 0 | 0 | 150.87 | 25.97 |

# Best Practices for Data Visualization

When creating data visualizations, it's important to follow best practices to ensure that your plots are clear and effective. These include making sure the data is visible, displaying patterns in the data clearly, being honest in portraying the magnitude of changes, and drawing your plot clearly without unnecessary stylistic components.

# End of Chapter Questions

1. Why is data visualization important in biostatistics?
2. Describe how to create a histogram and a scatter plot in Python. What type of data is each plot suitable for?
3. What are some best practices for data visualization? Give an example of a situation where not following these practices could lead to a misleading plot.
4. You are given a dataset containing measurements of various physical characteristics (e.g., height, weight, blood pressure) for a group of individuals. What types of plots might you create to explore this data, and why?

# Chapter 4: Describing data: Summary Statistics

Learning goals

- Effectively use mean and median to describe data.
- Explain the differences between the first and third quartiles.
- Calculate and interpret variance and standard deviations
- Identify the difference between numerical data and proportions
- Utilize Python to produce summary statistics

Summary statistics are integral to biostatistics as they provide a preliminary understanding of the data at hand. They offer an initial look into the central tendency, variability, and distribution of the data. In this chapter, we will cover mean, median, interquartile range, variance, standard deviation, proportions, and explore how to compute these summary statistics in Python.

## Mean

The mean, often referred to as the average, is the sum of all measurements divided by the number of measurements. It's a measure of central tendency that indicates where the data clusters around. The formula to calculate the sample mean ($\bar{y}$) is:

$$\bar{y} = \frac{\sum{y_i}}{n}$$

Σ denotes sum, yi refers to the ith observation, and n is the sample size.

In [1]:
```
x = [1, 4, 2, 3, 2]
n = len(x)

print("Mean of x =", round(sum(x)/n, 2))
```

```
Mean of x = 2.4
```

### Median

The median is the middle measurement in a **sorted** list of all measurements. If the list contains an even number of observations, the median is the average of the two middle numbers. Like the mean, it's a measure of central tendency, but unlike the mean, it's not affected by extreme values (outliers).

In [2]:
```
x = [1, 4, 2, 3, 2]
x.sort()

print("Median of x =", x[3])
```

```
Median of x = 3
```

## Video


Video link

### Interquartile Range

The Interquartile Range (IQR) represents the middle 50% of the measurements, i.e., the range between the first quartile (Q1) and the third quartile (Q3). It's a measure of statistical dispersion, indicating how spread out the measurements are.

In [46]:
```
x = [1, 4, 2, 3, 2]
x.sort()

pct25 = x[round((25 / 100) * (len(x) - 1))]
print('First quartile (25th percentile) =', pct25)

pct75 = x[round((75 / 100) * (len(x) - 1))]
print('Third quartile (75th percentile) =', pct75)
```

```
First quartile (25th percentile) = 2
Third quartile (75th percentile) = 3
```

### Variance and Standard Deviation

The variance ($s^2$) and standard deviation (s) measure the dispersion of measurements. They indicate how much the measurements deviate from the mean. Variance is the average squared deviation from the mean, while the standard deviation is the square root of the variance. They are calculated as follows:

Variance: $$s^2 = \frac{\Sigma(y_i - \bar{y})^2}{n - 1}$$

Standard Deviation: $$s = \sqrt{s^2}$$

> The numerator in the variance equation, $\Sigma(yi - \bar{y})^2$, is called the sum of squares, which forms the basis of most basic statistics.

In [49]:
```
import math
x = [1, 4, 2, 3, 2]

mean_x = sum(x) / len(x)
var_x = sum((xi - mean_x) ** 2 for xi in x) / (len(x) - 1)
sd_x = math.sqrt(var_x)

print('The variance of x =', round(var_x, 2))
print('The standard deviation of x =', round(sd_x, 2))
```

```
The variance of x = 1.3
The standard deviation of x = 1.14
```

## Video


Video link

### Proportions

In the context of categorical variables, proportions (p-hat) refer to the relative amount of a particular category in the dataset. The formula is:

$$ \hat{p} = \frac{x}{n}$$

> x is the number of times the category of interest occurs, and n is the total number of observations.

### Summary Statistics in Python

Python, with libraries such as pandas and NumPy, provides efficient ways to compute these summary statistics. Here we will take a look at measurements of penguin features from the `palmerpenguins` dataset. This is similar to the `iris` dataset we already worked with in the previous chapter.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

```
penguins = sns.load_dataset("penguins")

bill_length_mm = penguins['bill_length_mm'].dropna()

# Sample mean
mean = np.mean(bill_length_mm)
print("mean =", round(mean, 2))

# Sample standard deviation
std_dev = np.std(bill_length_mm, ddof=1)
print("standard deviation =", round(std_dev, 2))

# Sample size
n = len(bill_length_mm)
print("n =", n)


# Summary stats using pandas
print("Summary:\n", round(bill_length_mm.describe(), 2))
```

```
mean = 43.92
standard deviation = 5.46
n = 342
Summary:
 count    342.00
mean      43.92
std        5.46
min       32.10
25%       39.22
50%       44.45
75%       48.50
max       59.60
Name: bill_length_mm, dtype: float64
```

In the code above, we're importing necessary libraries and the Palmer Penguins dataset, selecting the 'bill_length_mm' column, and calculating the mean and standard deviation. We also used pandas' `describe()` function, which provides a convenient summary of various statistics (count, mean, standard deviation, minimum, 25th percentile (Q1), median, 75th percentile (Q3), maximum).

Note that when calculating the standard deviation with numpy, we pass in `ddof=1` to perform the calculation using n-1 in the denominator, which is the formula for the **sample standard deviation**. This makes the standard deviation a better estimate of the population parameter.

By understanding and effectively computing these summary statistics, you'll be able to start to understand your data, laying the groundwork for further biostatistical analysis.

# Practice problems

1. Calculate the mean of the following list using Python:

In [ ]:
```
x = [1, 4, 2, 3, 2]
```

2. Find the median of the given list using Python:

In [ ]:
```
x = [1, 4, 2, 3, 2]
```

3. Calculate the interquartile range (IQR) of the following list using Python:

In [ ]:
```
x = [1, 4, 2, 3, 2]
```

4. Compute the variance and standard deviation of the given list using Python:

In [ ]:
```
x = [1, 4, 2, 3, 2]
```

5. Calculate the proportion of a specific category in a dataset using Python:

In [ ]:
```
total_observations = 100
category_count = 30
```

# Chapter 5: Measuring Uncertainty with Standard Error and 95% Confidence Intervals

Learning Goals:

1. Understand the concepts of standard error and 95% confidence intervals.
2. Recognize the importance of standard error and confidence intervals in biostatistics.
3. Learn how to calculate standard error and 95% confidence intervals using Python.

## Introduction

In biostatistics, researchers often deal with uncertainty when estimating population parameters based on sample data. Standard error and confidence intervals are statistical measures that help quantify this uncertainty and provide a range within which the true population parameter is likely to fall. This chapter will introduce the concepts of standard error and 95% confidence intervals, explain their significance in biostatistics, and provide Python code examples for their calculation.

### Uncertainty in estimates

**Estimate**: an inference about a population parameter based on a subsample of that population. Since samples are only a subset of the entire population, there is a chance that estimates from repeated sampling

from that population can be off by a particular amount. This uncertainty can be visualized by producing a histogram of the **sampling distribution**.

## Video:


## Example: Body weight of penguins in Antarctica

There is natural variation in penguin size and weight in nature, which is influenced by factors such as species, age, sex, and environmental conditions. If we are interested in estimating the average penguin weight, we would need to take a subsample from the entire penguin population. However, the size of our sample can affect the accuracy estimate. A smaller sample size may not adequately represent the entire population, leading to a less reliable average estimate (i.e., increased error). Conversely, a larger sample size provides a more comprehensive representation of the population and reduces the potential for sampling bias (i.e., reduced error). By increasing the sample size, the estimate of the average penguin weight becomes more reliable, as it accounts for the natural variability observed in penguin populations and minimizes the impact of individual **outliers**, which area a small number of extremely small or extremely large individuals compared to the average.

We can illustrate the effect of sample size on the accuracy of our estimate by plotting the sampling distributions of either subsamples of 10 or subsamples of 50.

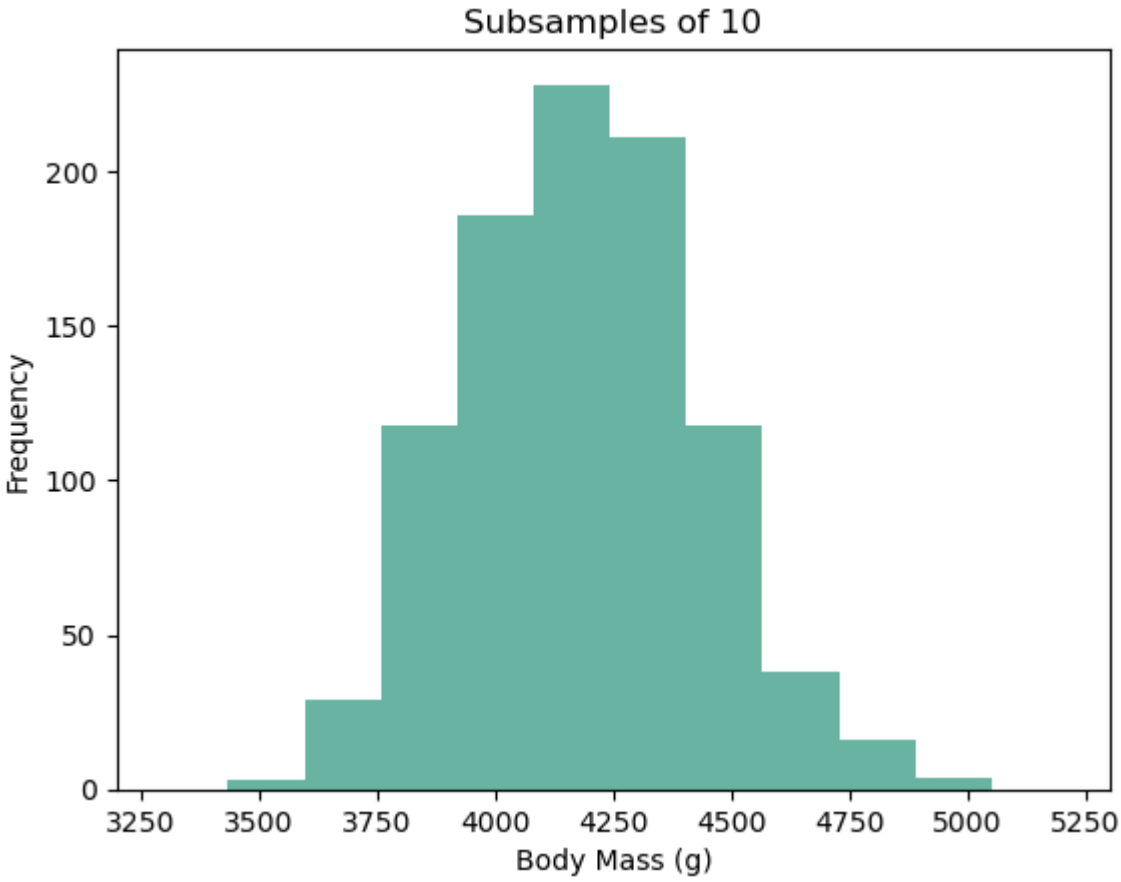
**Note**: These cells are separate but need to be run in order or they will not work.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Palmer Penguin dataset
df = sns.load_dataset("penguins")
```

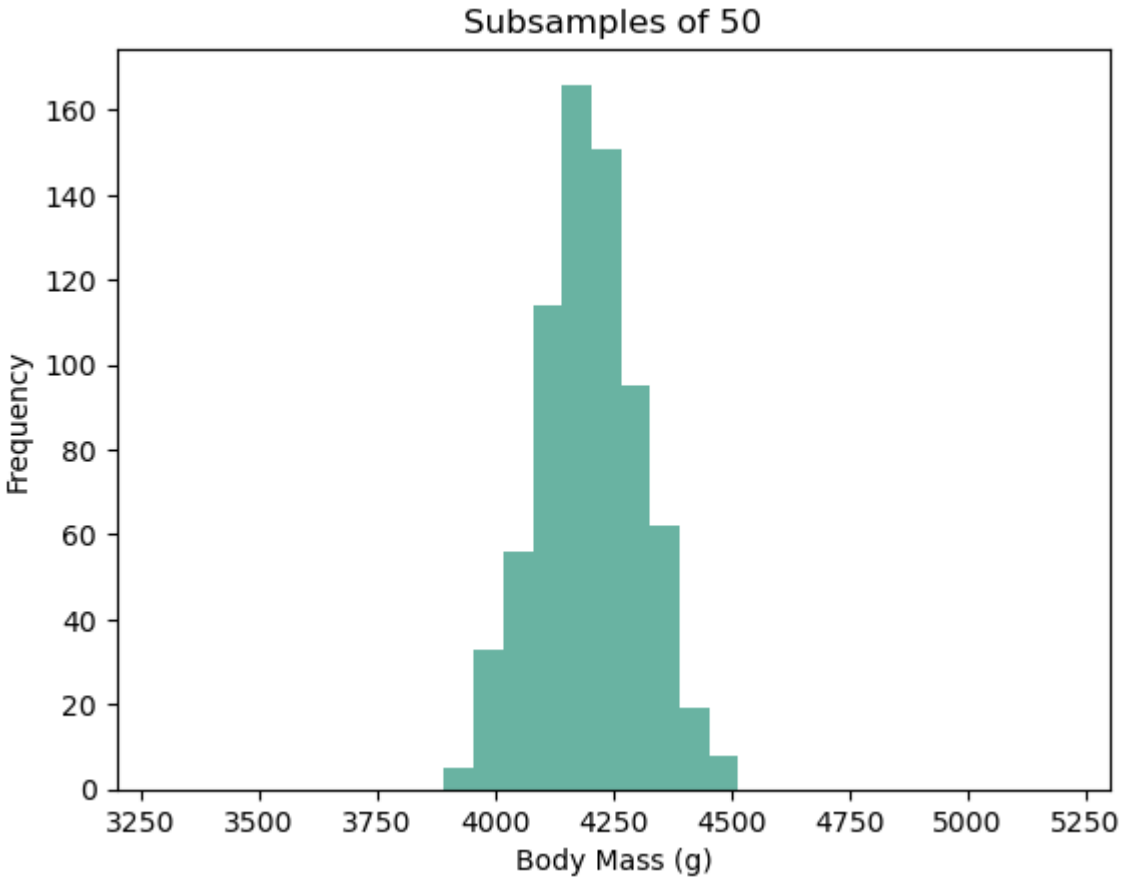
In [3]:
```
# Random subsample of body mass from 10 penguins
s_10 = np.random.choice(df.body_mass_g, size=10, replace=False)
print(s_10)
```

```
[4300. 3450. 5550. 2975. 4350. 3200. 3200. 3550. 3700. 3325.]
```

In [4]:
```
# Sampling distribution of 1000 subsamples of 10
dis_10 = []
for _ in range(1000):
    x = np.random.choice(df.body_mass_g, size=10, replace=False)
    dis_10.append(np.mean(x))

# Plotting the sampling distribution of subsamples of 10
plt.hist(dis_10, color="#69b3a2")
plt.xlim([3200, 5300])
plt.xlabel("Body Mass (g)")
plt.ylabel("Frequency")
plt.title("Subsamples of 10")
plt.show()
```




In [5]:
```
# Sampling distribution of subsamples of 50
dis_50 = []
for _ in range(1000):
    x = np.random.choice(df.body_mass_g, size=50, replace=False)
    dis_50.append(np.mean(x))

# Plotting the sampling distribution of 1000 subsamples of 50
plt.hist(dis_50, color="#69b3a2")
plt.xlim([3200, 5300])
plt.xlabel("Body Mass (g)")
plt.ylabel("Frequency")
plt.title("Subsamples of 50")
plt.show()
```




Notice that the histogram of the sampling distribution of the random subsample of 50 is skinnier the one for the subsample of 10. This illustrates that the sampling error is lower with a subsample of 50. This makes sense because the greater the sample (n), the lower the error due to random chance.

## Standard Error

The standard error (SE) measures the average amount of variability or dispersion of sample estimates around the true population parameter. It represents the precision of an estimate. A smaller standard error indicates a more precise estimate.

## Calculation of Standard Error

To calculate the standard error, use the following formula:

$SE_{\bar{y}} = \frac{s}{\sqrt{n}}$

In [7]:
```
import numpy as np
import seaborn as sns

# Load the Palmer Penguin dataset
df = sns.load_dataset("penguins")


def calculate_standard_error(data):
```

```
    sample_size = len(data)
    standard_deviation = np.std(data)
    standard_error = standard_deviation / np.sqrt(sample_size)
    return standard_error

sterr = calculate_standard_error(df['bill_length_mm'])
print(sterr)
```

```
0.2939303587811876
```

In this code example, `data` represents the sample data, and `calculate_standard_error` function returns the standard error.

## Confidence Intervals

A confidence interval (CI) is a range of values within which the true population parameter is likely to lie with a certain level of confidence. The most commonly used confidence level is 95%, which implies that if we repeated the sampling and estimation process many times, approximately 95% of the resulting confidence intervals would contain the true population parameter.

## Calculation of Confidence Intervals

To calculate the 95% confidence interval, use the following formula:

In [8]:
```
import seaborn as sns

def calculate_confidence_interval(data):
    sample_mean = np.mean(data)
    sample_size = len(data)
    standard_error = calculate_standard_error(data)
    margin_of_error = 2 * standard_error # The simplest calculation, using 2 * SE
    lower_bound = sample_mean - margin_of_error
    upper_bound = sample_mean + margin_of_error
    return (lower_bound, upper_bound)

# Load the Palmer Penguin dataset
df = sns.load_dataset("penguins")

ci_calc = calculate_confidence_interval(df['bill_length_mm'])
print(ci_calc)
```

```
(43.334069106999024, 44.50979054212378)
```

In this code example, `data` represents the sample data, and `calculate_confidence_interval` function returns a tuple containing the lower bound and upper bound of the confidence interval as 2 * SE.

## Video


**Note**: there are multiple ways to calculate 95% CI depending on your data and statistical methods.

## Standard Error and Confidence Intervals in Biostatistics

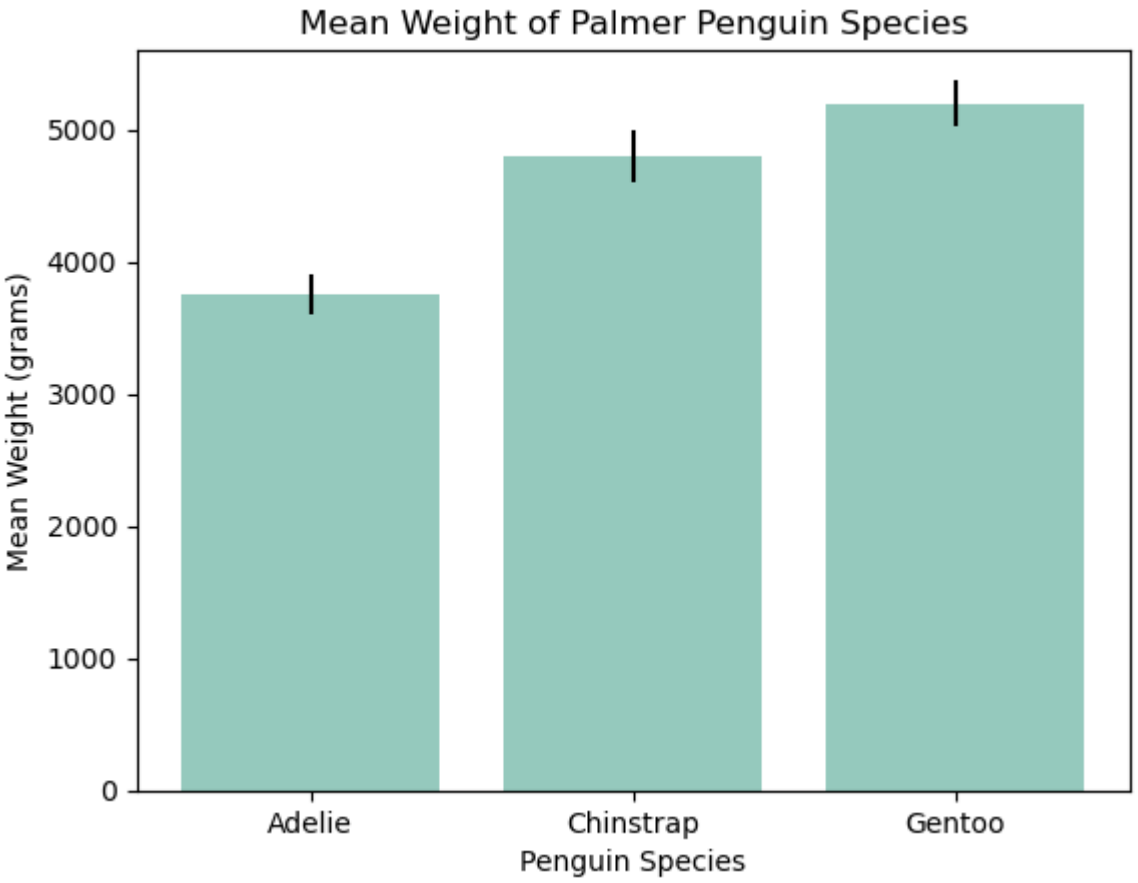
Standard error provides a measure of the precision of an estimate, allowing researchers to understand the reliability of their findings. It helps determine whether observed differences between groups or associations between variables are statistically significant. Standard error of the mean can be represented in text using the $\pm$ symbol.

Example, the average weight of a sample of 10 penguins is 4087 $\pm$ 223.8 g

Standard error of the mean are also important components to bar graphs which illustrate the means of multiple groups. The green bars represent the mean body weight of the three penguin species in the Palmer Penguin dataset, the black bars represent $\pm$ SE.

In [9]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Sample data of mean weights and corresponding standard errors for each penguin species
species = ['Adelie', 'Chinstrap', 'Gentoo']
mean_weights = [3750, 4800, 5200]  # Mean weights in grams
std_errors = [150, 200, 180]  # Standard errors of the mean weights

# Calculate the positions of the bars on the x-axis
x = np.arange(len(species))

# Plotting the bar graph
plt.bar(x, mean_weights, yerr=std_errors, color="#69b3a2", alpha=0.7)

# Customize the plot
plt.xlabel('Penguin Species')
plt.ylabel('Mean Weight (grams)')
plt.title('Mean Weight of Palmer Penguin Species')
plt.xticks(x, species)
```

```
plt.ylim([0, max(mean_weights) + max(std_errors) + 200])

# Display the plot
plt.show()
```

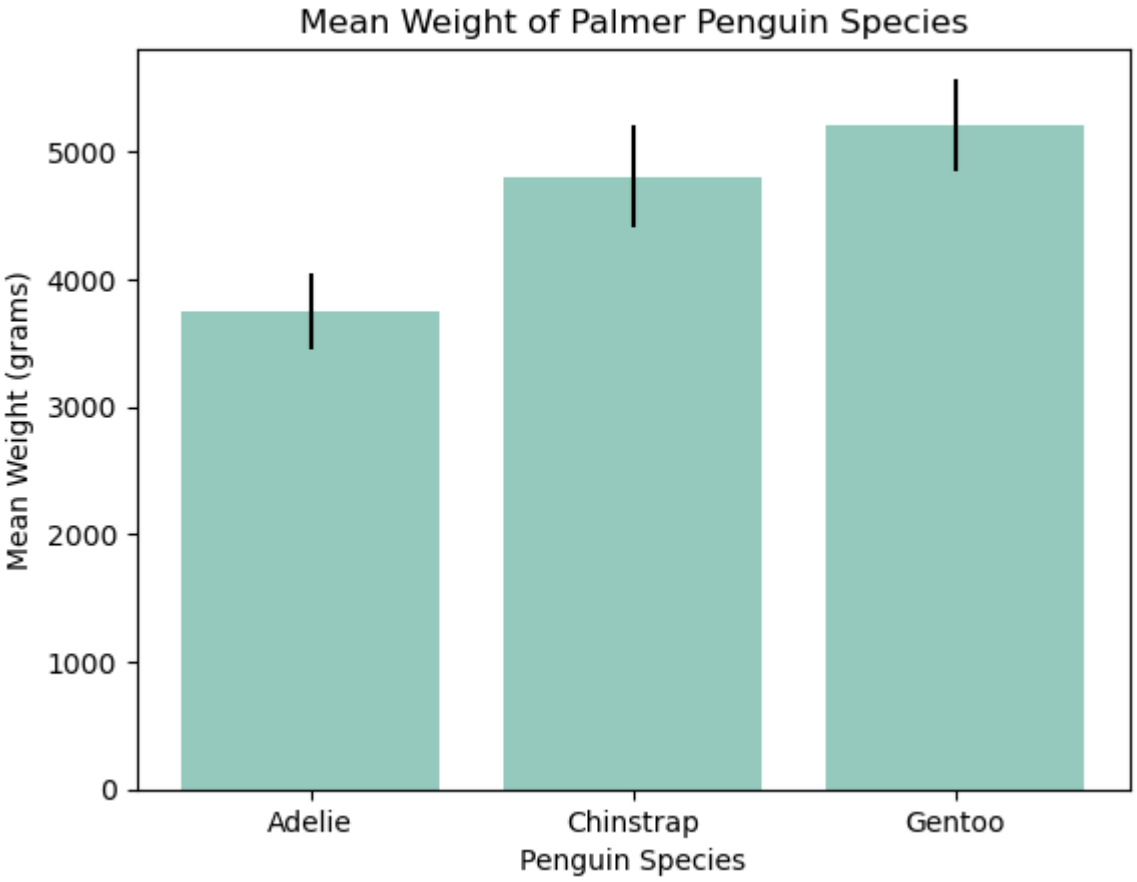



Confidence intervals provide a range of plausible values for the population parameter, which helps researchers communicate the uncertainty in their estimates. It allows readers to evaluate the practical significance of the findings and compare them to other studies. Like SE, 95% CI can be represented in bar plots and other graphical outputs.

In [10]:
```
import matplotlib.pyplot as plt
import numpy as np

# Sample data of mean weights and corresponding 95% confidence intervals for each penguin species
species = ['Adelie', 'Chinstrap', 'Gentoo']
mean_weights = [3750, 4800, 5200]  # Mean weights in grams
confidence_intervals = [300, 400, 360]  # 95% confidence intervals (2 * SE)

# Calculate the positions of the bars on the x-axis
x = np.arange(len(species))
```

In [11]:
```
# Plotting the bar graph
plt.bar(x, mean_weights, yerr=confidence_intervals, color="#69b3a2", alpha=0.7)

# Customize the plot
plt.xlabel('Penguin Species')
plt.ylabel('Mean Weight (grams)')
```

```
plt.title('Mean Weight of Palmer Penguin Species')
plt.xticks(x, species)
plt.ylim([0, max(mean_weights) + max(confidence_intervals) + 200])

# Display the plot
plt.show()
```




The difference between SE and 95% CI bars is that you can use the overlapping of the 95% CI to make statistical inferences. For example, since all of the 95% CI for the three penguin species do not overlap, then they are likely significantly different from each other.

**You cannot make this same claim with SE.**

## Practice problems

Using the data provided in the Palmer Penguin dataset

1. Calculate the standard error and 95% CI of the bill lengths of a random sample of 5 penguins
2. Calculate the standard error and 95% CI of the bill lengths of a random sample of 100 penguins
3. Are the SE and 95% CI different between #1 and #2? What is the explanation for the difference between the two?

# Chapter 6: Basic probability and Bayes' Theorem

Learning Goals:

1. Understand the fundamental concepts of probability theory.
2. Learn the principles of addition and multiplication in probability calculations.
3. Explore conditional probability and its application in real-world scenarios.
4. Introduce Bayes' Theorem and its role in updating probabilities.

## Introduction

Probability is a mathematical framework for quantifying uncertainty. It measures the likelihood of events occurring and provides a foundation for making informed decisions based on available information.

- Sample Space: The set of all possible outcomes of a random experiment.

- Event: A subset of the sample space representing a particular outcome or a combination of outcomes.
- Probability of an Event: A numerical value assigned to an event, ranging from 0 to 1, representing the likelihood of that event occurring.

## Probability symbology

P(A): Represents the probability of an event A occurring. For example, P(Head) represents the probability of getting a head in a coin toss.

P(A | B): Denotes the conditional probability of event A occurring given that event B has already occurred. For example, P(Rain | Cloudy) represents the probability of rain given that it is already cloudy.

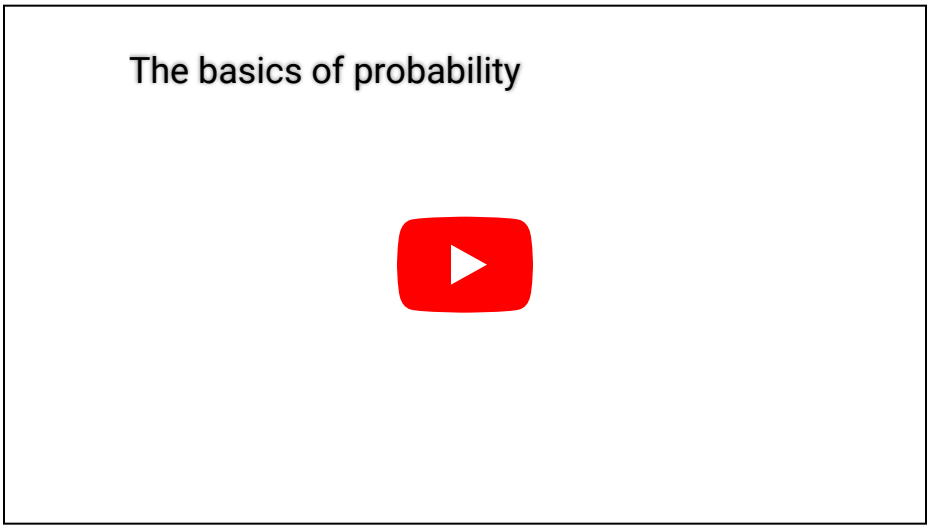
P(A ∩ B) or P(A and B): Represents the probability of both events A and B occurring simultaneously. This is called the intersection of events A and B.

P(A ∪ B) or P(A or B): Denotes the probability of either event A or event B or both occurring. This is called the union of events A and B.

P(A') or P(not A): Represents the probability of the complement of event A, i.e., the event A not occurring.

P(A,B): Denotes the joint probability of events A and B occurring together.

## Video



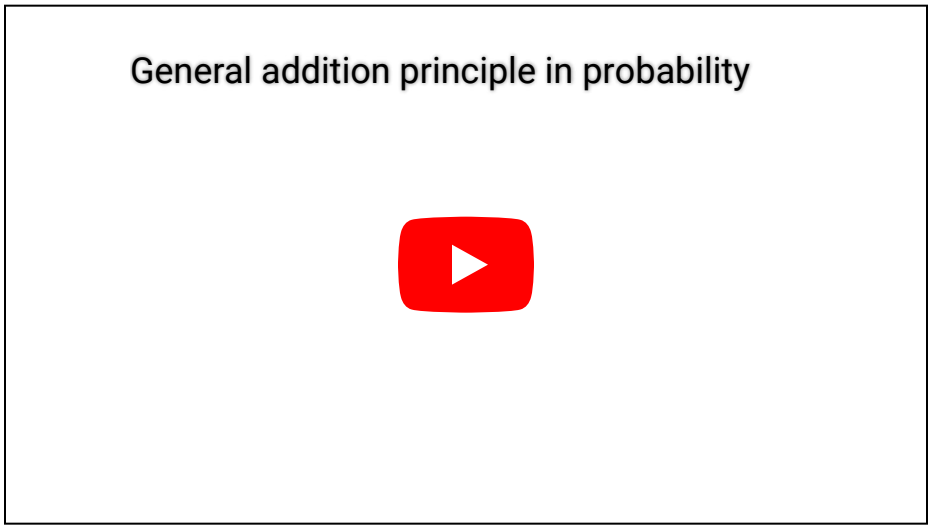

## Addition Rule

The addition rule states that the probability of the union of two events A and B is given by:

$$P(A ∪ B) = P(A) + P(B) - P(A ∩ B)$$
Example: Consider rolling a fair six-sided die. Let event A be rolling an even number (2, 4, or 6) and event B be rolling a number greater than 4 (5 or 6). The probability of event A is 3/6 (since there are three even numbers on the die), the probability of event B is 2/6, and the probability of their intersection is 1/6 (only

number 6 satisfies both events). Using the addition rule, we can calculate P(A ∪ B) as (3/6) + (2/6) - (1/6) = 4/6 = 2/3.

## Video

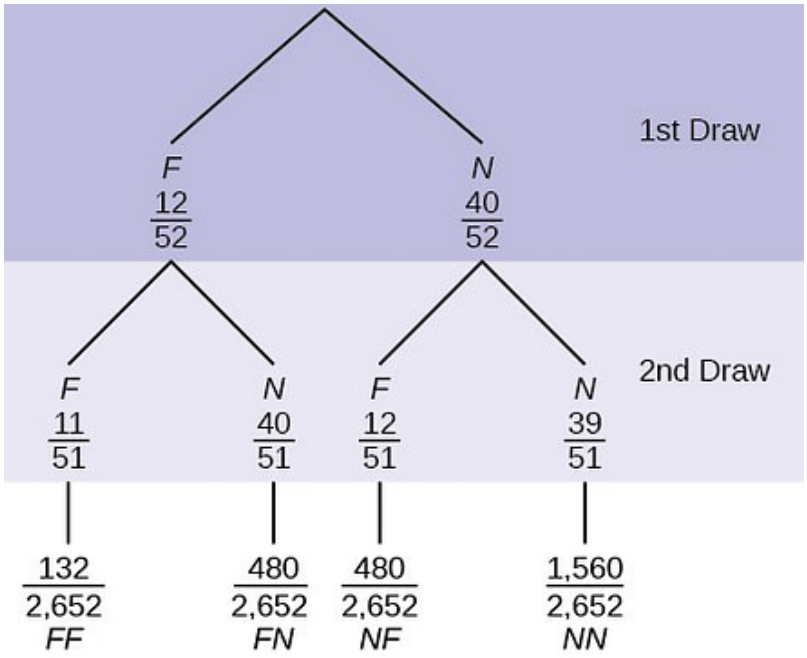



## Multiplication Rule

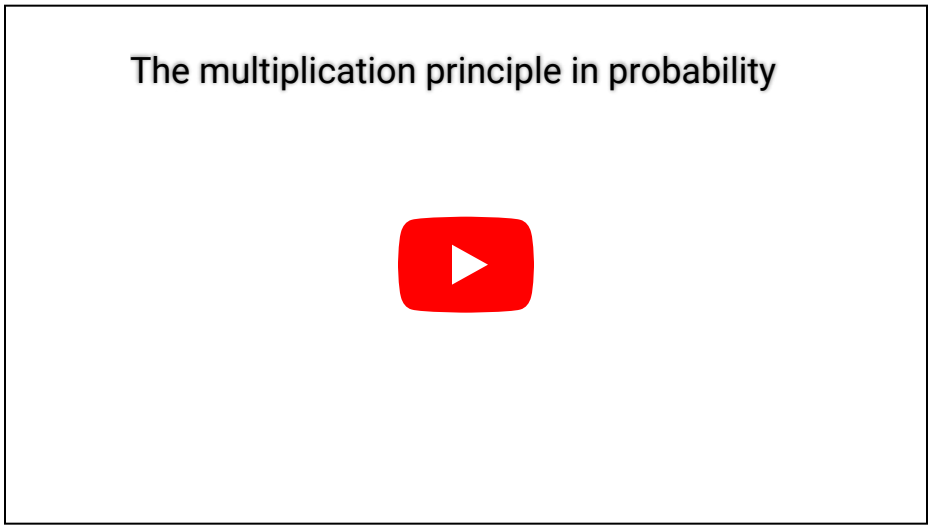
The multiplication rule states that the probability of the intersection of two independent events A and B is given by:

$$P(A ∩ B) = P(A) × P(B)$$
Example: Consider flipping a fair coin twice. Let event A be obtaining heads on the first flip, and event B be obtaining tails on the second flip. The probability of event A is 1/2 (since there are two equally likely outcomes: heads or tails), and the probability of event B is also 1/2. Since the flips are independent, we can calculate P(A ∩ B) as (1/2) × (1/2) = 1/4.

Probability trees (like the one below from Openstax.org) can be used to map out complex strings of probabilistic events.




## Video




## Conditional Probability

Conditional probability measures the probability of an event A occurring given that event B has already occurred. It is denoted as P(A|B) and calculated as: $$P(A|B) = P(A ∩ B) / P(B)$$

Example: Consider drawing two cards from a standard deck of 52 playing cards without replacement. Let event A be drawing a heart on the first card, and event B be drawing a heart on the second card. The probability of event A is 13/52 (since there are 13 hearts in a deck), and after drawing one heart, there are 12 hearts remaining out of the remaining 51 cards. Thus, the probability of event B given event A has occurred is 12/51. Therefore, P(A|B) = (13/52) × (12/51) = 1/17.

## Bayes' Theorem

Bayes' Theorem allows us to update the probability of an event based on new evidence. It is stated as follows:

$$P(A|B) = \frac{P(B|A) × P(A)}{P(B)}$$
Example: Consider a medical test for a rare disease, where only 1% of the population is affected. Let event A be having the disease, and event B be testing positive for the disease. Suppose the test is 95% accurate, meaning P(B|A) = 0.95, and the probability of having the disease P(A) = 0.01. If a randomly selected individual tests positive, we can calculate P(A|B) using Bayes' Theorem.

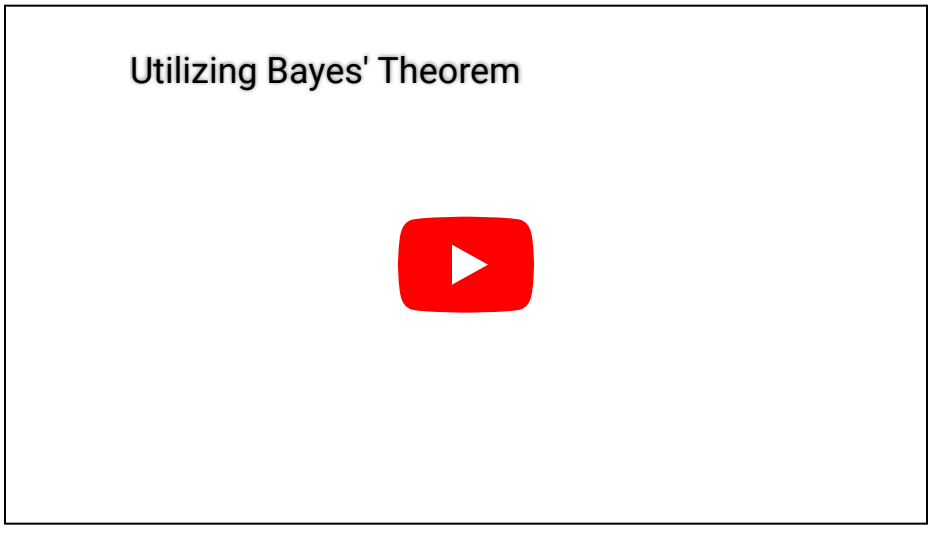
$$P(B) = P(B|A) × P(A) + P(B|not A) × P(not A)= (0.95 × 0.01) + (0.05 × 0.99) ≈ 0.0595$$
Now, applying Bayes' Theorem:

$$P(A|B) = \frac{(P(B|A) × P(A))}{P(B)} = \frac{(0.95 × 0.01)}{0.0595} ≈ 0.159$$
The implication of the result obtained using Bayes' Theorem in the given example is that even if an individual tests positive for a rare disease with a test that has a high accuracy rate (95% in this case), the probability of actually having the disease is relatively low (approximately 15.9% or 0.159).

This implies that a positive test result should not be interpreted as a definitive confirmation of the disease. Instead, further investigation, such as additional tests or medical evaluation, should be considered to confirm the diagnosis. In the context of rare diseases, false positives can be more common due to the low prevalence of the condition in the population. This phenomenon, known as the "base rate fallacy," emphasizes the importance of considering the prior probability (prevalence) of the disease when interpreting test results.

The implication of this result is that caution should be exercised in making decisions based solely on the outcome of a single test. Proper consideration of the accuracy of the test, as well as the prevalence of the disease in the population, is crucial to avoid misdiagnosis or unnecessary treatments. It highlights the need for a comprehensive approach that combines multiple pieces of information, including clinical evaluation and other diagnostic tools, to make informed decisions regarding an individual's health.

## Video



## Application of Probability Concepts

- Medical Testing: Using conditional probability to interpret the accuracy of medical tests. For example, assessing the probability of having a disease given a positive test result.
- Risk Assessment: Assessing the likelihood of certain events based on historical data. For instance, calculating the probability of an accident given weather conditions.
- Decision Making: Using probabilities to make optimal decisions under uncertainty. For instance, calculating the expected value to choose the most favorable option.

# End of chapter problems

1. Imagine that you have a population of 100 cats, 40 that are black with long fur, 30 that are white with short fur, 10 that are black with short fur, and 20 that are white with long fur.

- Are "black" and "long fur" mutually exclusive traits for this population of cats?
- Are "white" and "short fur" independent traits for this population of cats?

2. Consider a deck of cards

- What is the probability of drawing a king of any suit?
- What is the probability of drawing a face card that is also a spade?
- What is the probability of drawing a card without a number on it?
- What is the probability of drawing an ace?
- What is the probability of drawing a red ace?


3. When asked about their thoughts on the movie the Batman,

20% of movie-goers said that it was "excellent", 10% said that it was "pretty good"," 53% were "indifferent," 16% said that "it was pretty bad" and 1% said it was "especially bad". Only one answer per movie-goer was allowed.

- Are these five possible answers mutually exclusive? Explain.
- What is the probability that a movie-goer had a positive review of this movie?
- What is the probability that a movie-goer would review the movie as anything other than especially bad?

4. In a population of individuals, 45% are coffee drinkers. Suppose that coffee drinkers have a probability of 35% to develop insomnia during their lifetime. In contrast, non-coffee drinkers have a 10% lifetime chance of developing insomnia.

- What is the conditional probability of an individual from this population developing insomnia, given that they are a coffee drinker?

- Calculate the probability that a member of this population both drinks coffee and eventually develops insomnia.
- Using the general multiplication rule, calculate the probability that a member of this population both does not drink coffee and never develops insomnia.
- Calculate the probability that a person in this population would develop insomnia in their lifetime.
- Use Bayes' theorem to calculate the probability that a person from this population drank coffee, given that they eventually developed insomnia.

# Chapter 7: Hypothesis Testing and Experimental Design

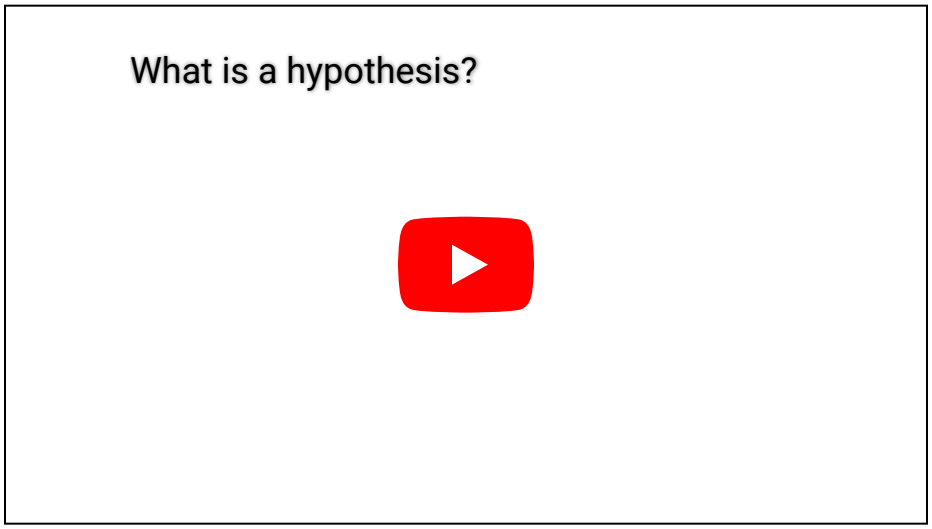
## Introduction

Hypothesis testing is the comparing observations to an expected null (or uninteresting) hypothesis. If our observations are different than what we would expect from the null, we reject that null.

- Null hypothesis (H0): The parameter of interest is zero.
- Alternative hypothesis (HA): The parameter of interest is not zero.

> We find evidence to reject the null, not accept the alternative (the possible alternatives are potentially infinite).

### Video




Video link

# Example hypotheses

$H_0: \bar{y} = 0$

$H_A: \bar{y} \neq 0$

## Test statistics and null distribution

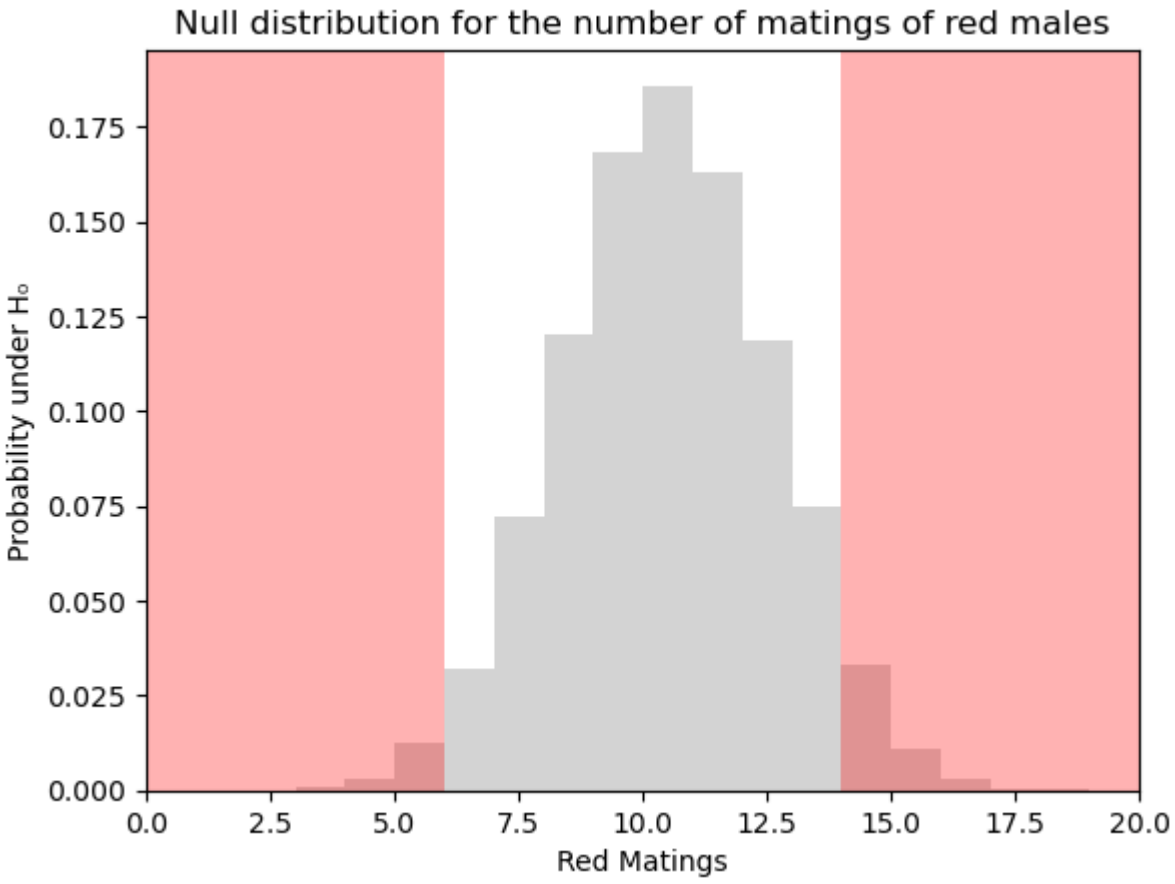
**Test statistics** are the values calculated from our observations to compare to the expected under the null.

The **null distribution** is the probability distribution of test statistics expected under the null

## Example null distribution

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)

fish_tank = np.array(["red"] * 100 + ["pale"] * 100)
red_mating_null = []

for i in range(10000):
    x = np.random.choice(fish_tank, size=20, replace=False)
    y = np.sum(x == "red")
    red_mating_null.append(y)

# Calculate histogram, but do not draw it
h, bins = np.histogram(red_mating_null, bins=range(21), density=True)
bin_centers = 0.5 * (bins[:-1] + bins[1:])

plt.figure()
plt.bar(bin_centers, h, width=1, color="lightgray")
```

```
plt.xlabel("Red Matings")
plt.ylabel("Probability under H₀")
plt.title("Null distribution for the number of matings of red males")
plt.xlim(0, 20)

# Add shaded regions for 2.5% tails
cutoff = np.percentile(red_mating_null, [2.5, 97.5])
plt.axvspan(0, cutoff[0], facecolor='red', alpha=0.3)
plt.axvspan(cutoff[1], 20, facecolor='red', alpha=0.3)

plt.show()
```




## P-values

**P-value**: the probability of obtaining our results given that the null is true.

We can calculate the P-value here by taking the sum of the area under the curve of our null distribution where our results and any more extreme results fall (and where the opposite extremes fall, because it is two-tailed).

A one-tailed test would calculate P using only the left or right of the null distribution.

## Video


Video link

## Decision rule

When working with P-value, we typically use a standard significance value, or $\alpha = 0.05$. Which means that we need to be 95% certain that our results are not due to random chance.

For the example above, $P = 0.017$, which is less than 0.05, therefore we reject the null that mating was random

## Errors

**Type I error**: rejecting a true null hypothesis. The probability of committing a Type I error = $\alpha$.

**Type II error**: failing to reject a false null hypothesis. A test has more power if there is a lower likelihood of committing a Type II error.

## Video

Video link

## Experimental design

Types of studies:

- Clinical study: two or more treatments are assigned to human subjects
- Laboratory study: two or more treatments are assigned to non-human subjects
- Field study: two or more treatments are assigned to non-human subjects in nature.

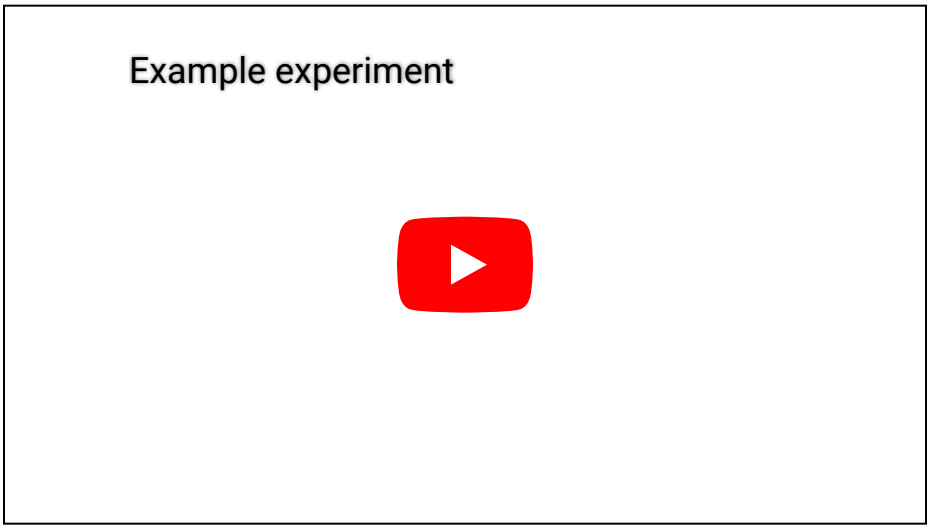
## Why well planed experiments are important

- Bias is reduced
  - Assigning **control groups**
  - Treatments assigned at **random**
  - **Blinding** to reduce conscious and unconscious bias

Sampling error is reduced - Treatment can be **replicated** multiple times - Experimental units can be **balanced** (same number of units per treatment) - Treatment replicates can be **blocked** together, to reduce effects of environmental variation

> Sometimes, observational studies must be done in lieu of an experimental study. While there is less opportunity for randomization, all other features of an experiment can be used.

## Video




**Blinding**, also referred to as "masking," is a method used in experimental research to reduce bias by preventing participants, researchers, or assessors from knowing certain key aspects of the study. The purpose of blinding is to minimize the potential for conscious or subconscious influence on the results or outcomes, thereby enhancing the objectivity and validity of the study.

There are different forms of blinding:

- Single-Blind: In a single-blind study, either the participants or the researchers/assessors are unaware of specific details. For example, in a clinical trial, participants might not know whether they are receiving the experimental treatment or a placebo, while the researchers who interact with the participants are aware of the assignments.
- Double-Blind: In a double-blind study, both the participants and the researchers/assessors are kept unaware of crucial information. This is often applied in clinical trials to avoid bias in both treatment administration and outcome assessment. For instance, neither the participants nor the researchers know who is receiving the actual treatment or the placebo.
- Triple-Blind: This involves a third party, such as a data analyst, being blinded to certain aspects of the study. This helps prevent any unintentional manipulation of the analysis or interpretation of results.

**Blocking** in experiments is a technique used to control and account for sources of variability or nuisance factors that could potentially affect the outcome of the study. It involves grouping similar experimental units together into blocks and then ensuring that each treatment group is represented within each block. This helps to minimize the impact of confounding variables and enhance the accuracy and validity of the experimental results.

# End of chapter questions

1. A pharmaceutical company is testing a new drug for reducing blood pressure. In a randomized controlled trial, they administer the drug to one group and a placebo to another. After analyzing the data, they calculate a p-value of 0.03. What does this p-value indicate about the results of the experiment? How would you interpret this value in terms of the null and alternative hypotheses? Make sure to include the appropriate hypotheses.

2. In a biodiversity study of a remote rainforest ecosystem, researchers use a technological sampling method to detect the presence of a rare species of bird. They have noted that their sampling technique has a 15% false positive rate, meaning it sometimes identifies the species even when it's not present in a sampled area. Explain the concepts of Type I and Type II errors in this ecological context. Provide an example of each error using the bird species study. How could the researchers adjust their sampling approach to minimize these errors while considering the conservation implications of the species' rarity?

3. Researchers are investigating the effects of a new exercise program on weight loss. They recruit participants by posting flyers at local gyms, but they notice that most of the participants are already physically active. How could this lead to bias in the study results? Suggest two strategies the researchers could use to reduce this bias and ensure a more representative sample.

4. A botanist aims to investigate the effects of varying light intensities on the growth of a specific plant species. Outline three crucial factors the botanist should take into account when designing this experiment. How might randomization, blinding, and controlling confounding variables contribute to the reliability of the study's conclusions? Provide concrete examples for each factor.

# Chapter 8: Binomial Distribution and Binomial Test

Learning Goals:

1. Understand the binomial distribution and its characteristics.
2. Learn how to calculate probabilities using the binomial distribution.
3. Introduce the binomial test and its application in hypothesis testing.
4. Utilize Python code examples to calculate probabilities, perform the binomial test, and visualize the binomial distribution.

## Introduction

The binomial distribution models the number of successes in a fixed number of independent Bernoulli trials, where each trial has the same probability of success. It is characterized by two parameters: the number of trials (n) and the probability of success in each trial (p).

### Video


Video link

### Characteristics of the Binomial Distribution

- Probability Mass Function (PMF): The function that gives the probability of obtaining a specific number of successes.
- Mean and Variance: The mean of the binomial distribution is given by np, and the variance is np(1-p).
- Shape: The binomial distribution is usually right-skewed for small p or left-skewed for large p.

## Calculating probabilities using the binomial distribution

Probability Mass Function (PMF) is given by the formula:

$$P(X = k) = C(n, k) * p^k * (1-p)^{(n-k)}$$
Example: Consider flipping a fair coin 10 times. Let's calculate the probability of getting exactly 7 heads using the binomial distribution. Here, n = 10 (number of trials) and p = 0.5 (probability of heads).

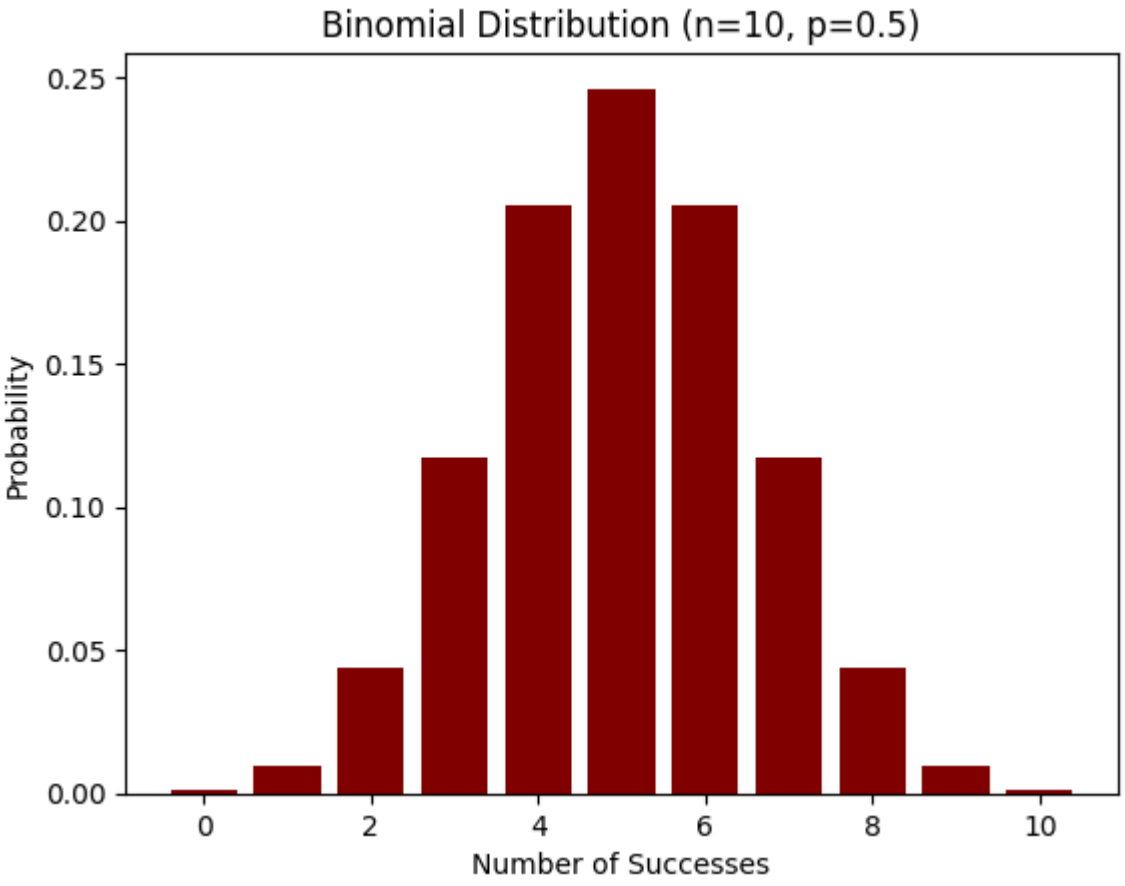
$$P(X = 7) = C(10, 7) * 0.5^7 * (1-0.5)^{(10-7)}$$

## Cumulative Distribution Function (CDF)

The CDF of the binomial distribution gives the probability of obtaining up to a certain number of successes.

$$P(X ≤ k) = \sum{P(X = i) }$$
$$\text{for i = 0 to k}$$

In [1]:
```
from scipy.stats import binom
import matplotlib.pyplot as plt

n = 10
p = 0.5

x = range(n+1)
y = [binom.pmf(k, n, p) for k in x]

plt.bar(x, y, color = 'maroon')
plt.xlabel("Number of Successes")
plt.ylabel("Probability")
plt.title("Binomial Distribution (n=10, p=0.5)")
plt.show()
```




## Video


Video link

## Calculating probabilities using the binomial distribution in Python

In [2]:
```
from scipy.stats import binom

n = 10
p = 0.5
```

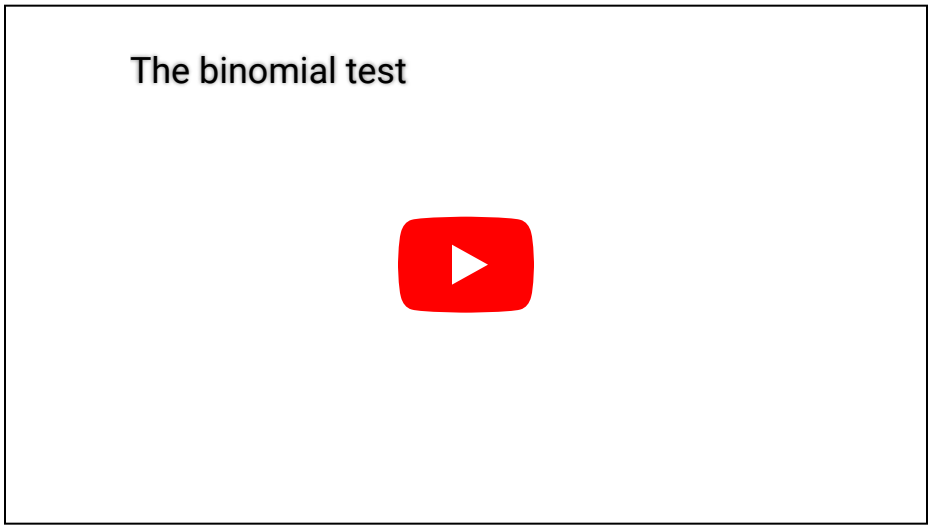
```
k = 7

probability = binom.pmf(k, n, p)
print("Probability of getting exactly 7 heads:", probability)
```

```
Probability of getting exactly 7 heads: 0.11718749999999999
```

## The Binomial Test

The binomial test is a statistical test used to assess whether the observed number of successes significantly deviates from an expected value under the null hypothesis. It is particularly useful for binary data or situations with a fixed number of trials.

## Video




Video link

## Performing the Binomial Test

In the binomial test, we specify the null hypothesis (H0) and alternative hypothesis (H1) and compare the observed number of successes to the expected value under H0. The p-value is calculated based on the binomial distribution, and if it falls below a predetermined significance level, we reject the null hypothesis.

In [3]:
```
from scipy.stats import binomtest

observed_successes = 7
total_trials = 10
expected_successes = 5

results = binomtest(observed_successes, total_trials, expected_successes/total_trials)
print("p-value:", round(results.pvalue, 3))
```

```
p-value: 0.344
```

## End of chapter problem

In some species of birds, juveniles help other adult birds to raise offspring. Scientists are unsure what the evolutionary advantage of this sort of altruism is in the wild, however, it is postulated that these juveniles are likely to be preferentially helping adults that they are more closely related to (kin selection). To test this, 20 birds that each lived nearby two nests of adults, one with relatives, the other with non-relatives. It was observed that 16 of the birds helped their kin, while 4 helped the non-kin.

Is there preferential care?

In [ ]:

# Chapter 9: $\chi^2$ Test for Independence and Goodness-of-Fit Test

Learning Goals:

1. Understand the concept and applications of contingency tables.
2. Learn how to create and interpret contingency tables.
3. Explore the $\chi^2$ test for independence.
4. Utilize Python code examples to perform analysis on contingency tables.
5. Understand the $\chi^2$ Goodness-of-Fit Test and its purpose in statistical analysis.
6. Learn how to perform the $\chi^2$ Goodness-of-Fit Test to assess the fit between observed and expected frequencies.
7. Utilize Python code examples to conduct the $\chi^2$ Goodness-of-Fit Test and interpret the results.

# Contingency Tables and $\chi^2$ Test for Independence

## Contingency tables

Contingency tables, also known as cross-tabulation or crosstab, are used to summarize and examine the relationship between two categorical variables. They display the frequency distribution of the variables across different categories.

## Applications of Contingency Tables

Contingency tables are commonly used in various scenarios, including:

- Analyzing survey data.
- Investigating association between variables.
- Assessing the distribution of data across categories.

## Creating and Interpreting Contingency Tables

A contingency table organizes data into rows and columns, representing the categories of the two categorical variables being studied. The cells of the table contain the frequency or count of observations that fall into each combination of categories.

Contingency tables allow us to observe patterns and relationships between the categorical variables. We can examine the distribution of frequencies across the cells to identify any associations or dependencies.

Example contingency table:

| | Group1 | Group2 |
|---|---|---|
| Outcome1 | 15 | 5 |
| Outcome2 | 3 | 16 |

## $\chi^2$ Test for Independence

The $\chi^2$ test for independence is used to determine whether there is a statistically significant association between two categorical variables. It compares the observed frequencies in a contingency table with the frequencies that would be expected if the variables were independent.

In [ ]:
```
import numpy as np
import scipy.stats as stats
```

```
# Example data
observed = np.array([[15, 5],
                     [3, 16]])

# Perform chi-squared test for independence
chi2, p_value, dof, expected = stats.chi2_contingency(observed)
print("Chi-Squared Test Results:")
print("Chi-squared statistic:", round(chi2, 2))
print("p-value:", round(p_value, 2))
print("Degrees of Freedom:", round(dof, 2))
print("Expected Frequencies:")
print(expected)
```

```
Chi-Square Test Results:
Chi-square statistic: 11.47
p-value: 0.0
Degrees of Freedom: 1
Expected Frequencies:
[[ 9.23076923 10.76923077]
 [ 8.76923077 10.23076923]]
```

# Interpreting $\chi^2$ Statistic and p-value

- $\chi^2$ statistic: Measures the overall association between variables.
- p-value: Determines the statistical significance of the association.
- If the p-value is below a predetermined significance level (e.g., 0.05), we reject the null hypothesis of independence and conclude that there is evidence of an association.

# $\chi^2$ Goodness-of-Fit Test

The $\chi^2$ Goodness-of-Fit Test is a statistical test used to determine if an observed sample fits a specified theoretical distribution. It compares the observed frequencies with the expected frequencies based on the hypothesized distribution.

## Applications of the $\chi^2$ Goodness-of-Fit Test

- Testing Hypotheses: Assessing whether observed data follows a specific distribution.
- Model Evaluation: Checking if a theoretical model adequately represents observed data.

## Performing the $\chi^2$ Goodness-of-Fit Test

The $\chi^2$ statistic measures the discrepancy between observed and expected frequencies. It is calculated using the formula:

$$\chi^2 = \frac{(O - E)^2}{E}$$

**O** represents the observed frequencies, **E** represents the expected frequencies based on the

hypothesized distribution, and the sum is taken over all categories or bins.

## Degrees of Freedom

The degrees of freedom (df) for the $\chi^2$ Goodness-of-Fit Test depend on the number of categories and constraints imposed by the test. For the test of fit to a specified distribution, df = number of categories - 1.

## P-value and Interpretation

The p-value associated with the $\chi^2$ Goodness-of-Fit Test represents the probability of obtaining the observed data or more extreme results if the null hypothesis (goodness-of-fit) is true. A smaller p-value suggests stronger evidence against the null hypothesis.

## Assumptions and Limitations of the Chi-Squared Goodness-of-Fit Test

Assumptions

- Independence: The observations within each category or bin are assumed to be independent.
- Expected Frequencies: The expected frequencies should be sufficiently large (typically at least 5) to satisfy the asymptotic properties of the $\chi^2$ distribution.

Limitations

- Sample Size: The $\chi^2$ Goodness-of-Fit Test may not be reliable with small sample sizes.
- Cell Frequency: If any cell in the contingency table has an expected frequency below a certain threshold (e.g., 5), the $\chi^2$ test results may be unreliable.

## Video


Video link

# Performing the Chi-Squared Goodness-of-Fit Test

In [3]:
```
from scipy.stats import chisquare

observed = [10, 15, 25, 20]  # Observed values
expected = [12, 15, 20, 23]  # Expected values based on hypothesized distribution

chi2, p_value = chisquare(f_obs=observed, f_exp=expected)
print("Chi-Squared Statistic:", round(chi2, 2))
print("p-value:", round(p_value, 2))
```

```
Chi-Squared Statistic: 0.95
p-value: 0.81
```

## Interpreting the Results

Compare the p-value to a predetermined significance level (e.g., 0.05) to determine the statistical significance. If the p-value is below the significance level, we reject the null hypothesis and conclude that the observed data does not fit the hypothesized distribution.

# Chi-squared in biology

Gregor Mendel developed the principle of independent assortment, which states that alleles will segregate independently into gametes in a dihybrid with a phenotypic ratio of **9:3:3:1**. For example, see this dihybrid cross table of a BbEe x BbEe hybrid fly. The `B` gene codes for body color and the `E` gene codes for eye color. The dominant body color is brown, and the recessive color is black, the dominant eye color is red and the recessive eye color is brown. See that there are a 9:3:3:1 ratio of possible offspring. All possible combinations are represented in the table below.

| | **BE** | **Be** | **bE** | **be** |
|---|---|---|---|---|
| BE | BEBE | BEBe | BEbE | BEbe |
| Be | BeBE | BeBe | BebE | Bebe |
| bE | bEBE | bEBe | bEbE | bEbe |
| Be | beBE | beBe | bEbE | bebe |

However, we now know that sometimes certain alleles are associated with each other, or in other words are not independent. We can test to see if alleles are linked using the $\chi^2$ goodness of fit test.

Let's say we take flies that are heterozygous for both the B and E genes. We perform `100` dihybrid crosses and get the following offspring phenotypes:

| **Genotype** | **# individuals** |
|---|---|
| Brown, red | 50 |

| Genotype | # individuals |
|---|---|
| Brown, brown | 20 |
| Black, red | 20 |
| Black, brown | 10 |
| | Total = 100 |

We can use the chi-square test to see if these genes independently assort. If we expect a 9:3:3:1 ratio, we can calculate the expected phenotypes of the offspring:

| Genotype | # individuals |
|---|---|
| Brown, red | 9/16 * 100 = 56.25 |
| Brown, brown | 3/16 * 100 = 18.75 |
| Black, red | 3/16 * 100 = 18.75 |
| Black, brown | 1/16 * 100 = 6.25 |
| | Total = 100 |

In [6]:
```python
from scipy.stats import chisquare

observed = [50, 20, 20, 10]  # Observed values
expected = [56.25, 18.75, 18.75, 6.25]  # Expected values based on hypothesized distribution

chi2, p_value = chisquare(f_obs=observed, f_exp=expected)
print("Chi-Squared Statistic:", round(chi2, 2))
print("p-value:", round(p_value, 2))
```

```
Chi-Squared Statistic: 3.11
p-value: 0.37
```

# End of chapter question

Let's suppose that in aardvarks, front foot hair length is controlled by a single gene with two alleles, and let's call them `A` for the dominant allele and `a` for the recessive allele. In individuals have at least one `A` allele, they have a long front foot hair, but if they have both `a` alleles, they have a shorter foot hair.

We take a random sample of 230 aardvarks in a national park in Africa and genotype them. We find that 43 are `AA`, 32 are `Aa`, and 10 are `aa`. If having the short hair allele does not impact fitness (survival and reproduction),then individuals with 0, 1, or 2 copies of the "b" allele should follow a binomial distribution.

Test this using the $\chi^2$ goodness of fit test.

In [ ]:

# Chapter 10: Normal Distribution and Parametric Tests

Learning Goals:

1. Understand the properties and characteristics of the normal distribution.
2. Learn about parametric tests and their applications.
3. Explore the one-sample t-test, paired t-test, and two-sample t-test.
4. Understand the assumptions underlying parametric tests and how to assess them.
5. Utilize Python code examples to perform parametric tests and assess assumptions.

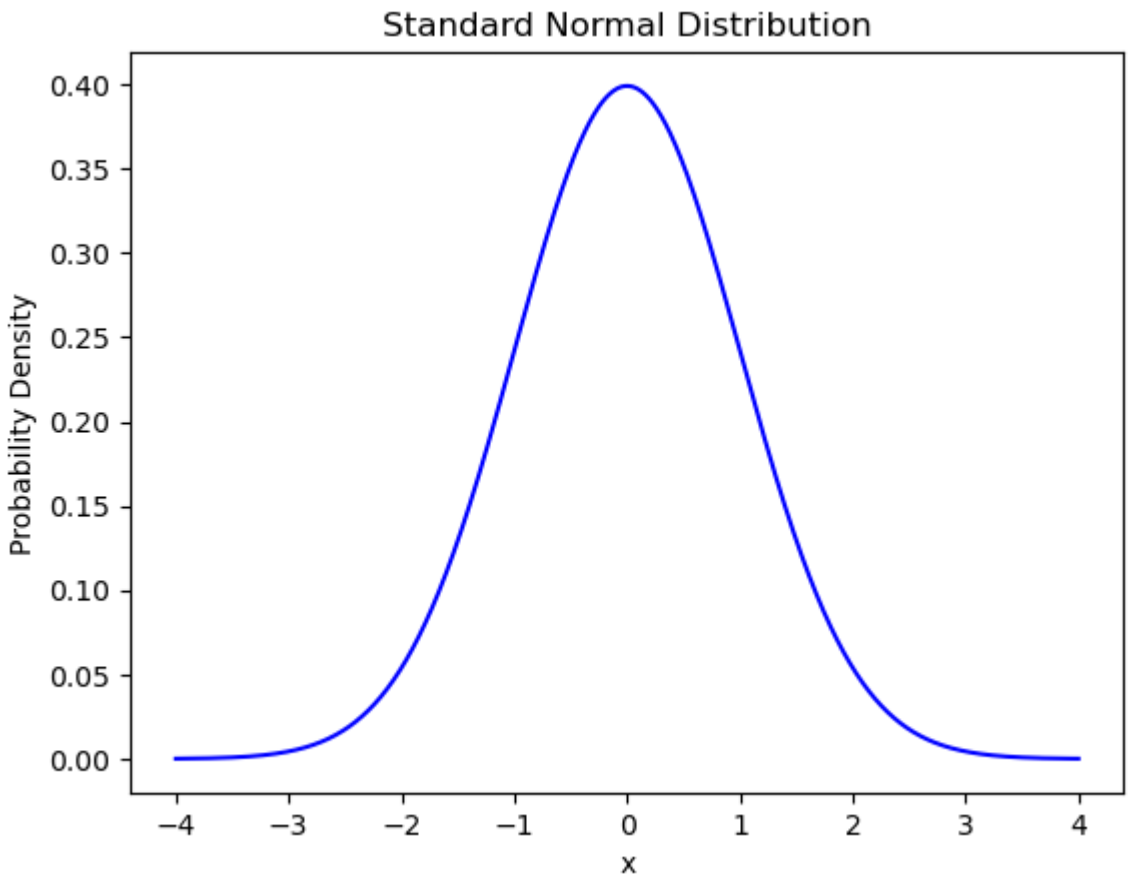
## Introduction

The normal distribution, also known as the Gaussian distribution, is a continuous probability distribution that is symmetric and bell-shaped. It is characterized by its mean and standard deviation.

### Standard Normal Distribution

The standard normal distribution has a mean of 0 and a standard deviation of 1. It serves as a reference distribution for statistical calculations.

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Generate values for x-axis
x = np.linspace(-4, 4, 1000)

# Calculate corresponding y-values from the standard normal distribution
y = (1 / np.sqrt(2 * np.pi)) * np.exp(-0.5 * x**2)

# Plot the standard normal distribution
plt.plot(x, y, color='blue')
plt.xlabel('x')
plt.ylabel('Probability Density')
plt.title('Standard Normal Distribution')
plt.grid(False)
plt.show()
```

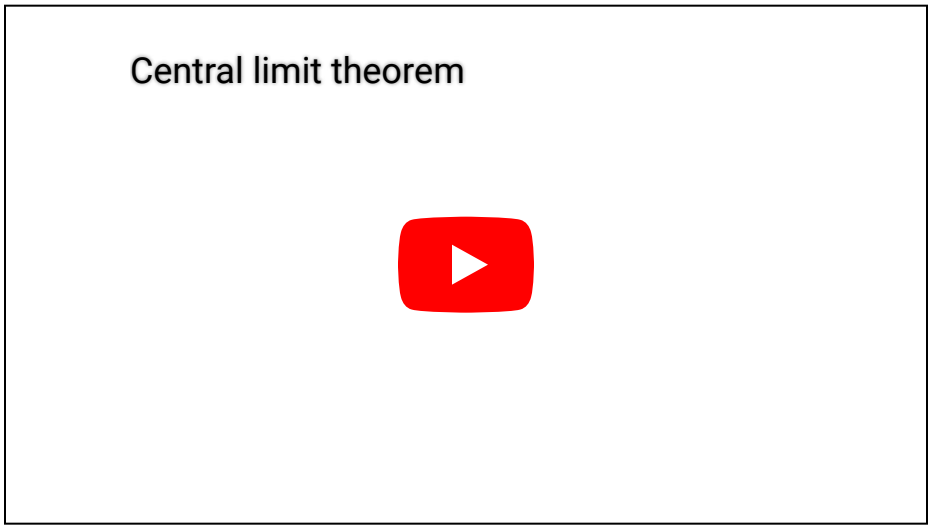



## Video


## Utilities of the normal distribution

The normal distribution is extremely valuable in statistics due to its universal applicability, enabled in part by the **Central Limit Theorem**, which states that sample means tend to follow a normal distribution, regardless of the original population distribution. This is critical to statistical methods like hypothesis testing and confidence intervals. The distribution's simplicity, characterized by the mean and standard deviation, facilitates mathematical analysis and description of real-world phenomena. Serving as an approximation for diverse distributions and providing standardized scores for comparison, the normal distribution simplifies calculations and supports inference about population parameters. Its prevalence in fields such as process control, quality assurance, and population studies further underscores its indispensability, making it a foundational concept for statistical analysis.

## Video




# Parametric Tests and their Applications

Parametric tests are statistical tests that make assumptions about the underlying distribution of the data. These tests are based on specific parameters, such as means or variances, and are useful for comparing groups or testing hypotheses.

## Applications of Parametric Tests

Parametric tests are commonly used in various scenarios, including:

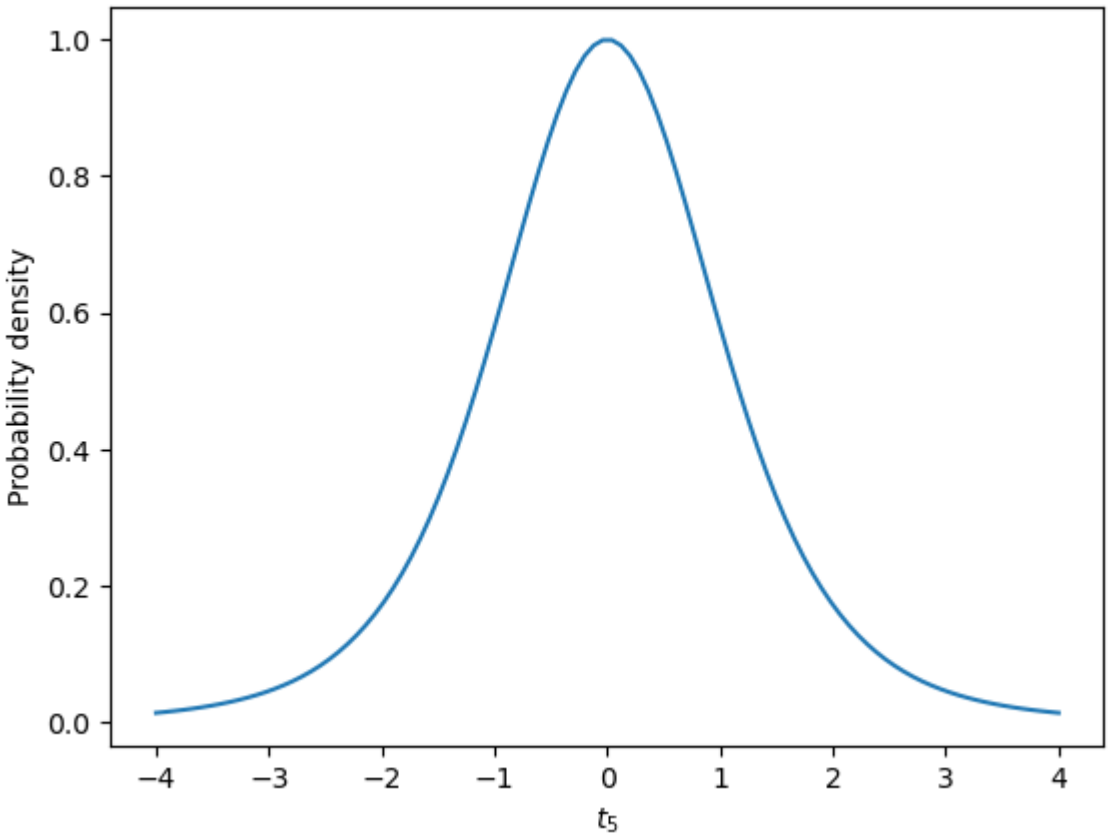
- Comparing a sample mean to a known or hypothesized value.
- Comparing two related samples.
- Comparing two independent samples.

Parametric tests are based on the normal distribution which are based on population means ($\mu$) and standard deviations ($\sigma$). In real-life studies, however, we rarely know the population standard deviation. Therefore, we need to estimate it using the standard error of the mean (SE). This then changes our normal sampling distribution to a **t-distribution**.

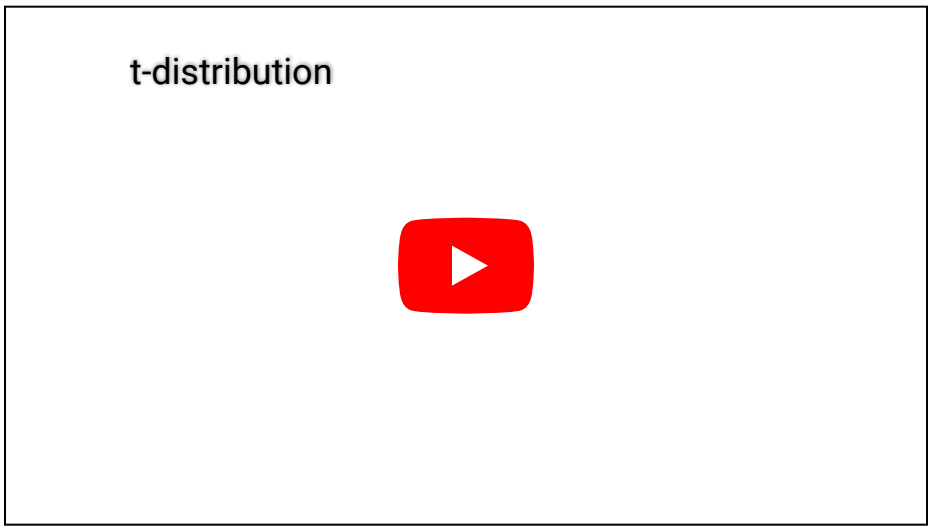
$t = \frac{\bar{x} - \mu_0}{{SE}_{\bar{y}}}$

The t distribution incorporates sampling error, and therefore is wider than a standard normal. Because we need to estimate $\sigma$ before calculating t, a t-distribution will have a degrees of freedom of `n - 1`. The one below is a t distribution with df = 5.

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt

def dt(x, df):
    return (1 + x**2 / df)**(-((df + 1) / 2))

x = np.linspace(-4, 4, 100)
y = dt(x, df=5)

plt.plot(x, y)
plt.xlabel('$t_5$')
plt.ylabel('Probability density')
plt.show()
```




## Video




# One-Sample t-test

The one-sample t-test is used to determine if the mean of a single sample significantly differs from a hypothesized value.

$H_0 = \mu_1 = \mu_0$

$H_A = \mu_1 \neq \mu_0$

$t = \frac{\bar{y} - \mu_0}{SE_{\bar{y}}}$

## Assumptions for the One-Sample t-test

- The data are normally distributed.
- The observations are independent.
- The population standard deviation is unknown.

## Video


## Performing the One-Sample t-test in Python

In [13]:
```python
from scipy.stats import ttest_1samp

# Example data
data = [4.8, 5.2, 5.1, 4.9, 5.3, 5.0, 5.2, 5.1]

# Hypothesized mean
hypothesized_mean = 5.0

# Perform one-sample t-test
t_statistic, p_value = ttest_1samp(data, hypothesized_mean)
print("One-Sample t-test Results:")
print("t-statistic:", round(t_statistic, 2))
print("p-value:", round(p_value, 2))
```

```
One-Sample t-test Results:
t-statistic: 1.27
p-value: 0.24
```

# Paired t-test

The paired t-test is used to compare the means of two related samples, typically collected under different conditions or time points.

$H_0 = \mu_d = 0$

$H_A = \mu_d \neq 0$

$t = \frac{\bar{d} - \mu_{d0}}{SE_{\bar{d}}}$

## Assumptions for the Paired t-test

- The differences between paired observations are normally distributed.
- The differences are independent.
- The population standard deviation of the differences is unknown.

## Performing the Paired t-test in Python

In [14]:
```
from scipy.stats import ttest_rel

# Example data
before = [18, 19, 20, 17, 16]
after = [15, 17, 19, 16, 16]

# Perform paired t-test
t_statistic, p_value = ttest_rel(before, after)
print("Paired t-test Results:")
print("t-statistic:", round(t_statistic,2 ))
print("p-value:", round(p_value, 2))
```

```
Paired t-test Results:
t-statistic: 2.75
p-value: 0.05
```

# Two-Sample t-test

The two-sample t-test is used to compare the means of two independent samples.

$H_0 = \mu_1 = \mu_2$

$H_A = \mu_1 \neq \mu_2$

$t = \frac{\bar{y}_1 - \bar{y}_2}{SE_{\bar{y}_1 - \bar{y}_2}}$

where

$SE_{\bar{y}_1 - \bar{y}_2} = \sqrt{s^{2}_{p}(\frac{1}{n_1} + \frac{1}{n_2})}$

where

$s^{2}_{p} = \frac{(n_1 - 1)s^{2}_{1} + (n_2 - 1)s^{2}_{2}}{n_1 + n_2 - 2}$

## Assumptions for the Two-Sample t-test

- The data in each group are normally distributed.
- The observations are independent.
- The population standard deviations of the two groups are equal.

## Performing the Two-Sample t-test in Python

In [15]:
```
from scipy.stats import ttest_ind

# Example data
group1 = [15, 18, 20, 22, 17]
group2 = [12, 14, 16, 15, 18]

# Perform two-sample t-test
t_statistic, p_value = ttest_ind(group1, group2)
print("Two-Sample t-test Results:")
print("t-statistic:", round(t_statistic, 2))
print("p-value:", round(p_value, 2))
```

```
Two-Sample t-test Results:
t-statistic: 2.17
p-value: 0.06
```

# Assessing Assumptions

## Normality Assumption

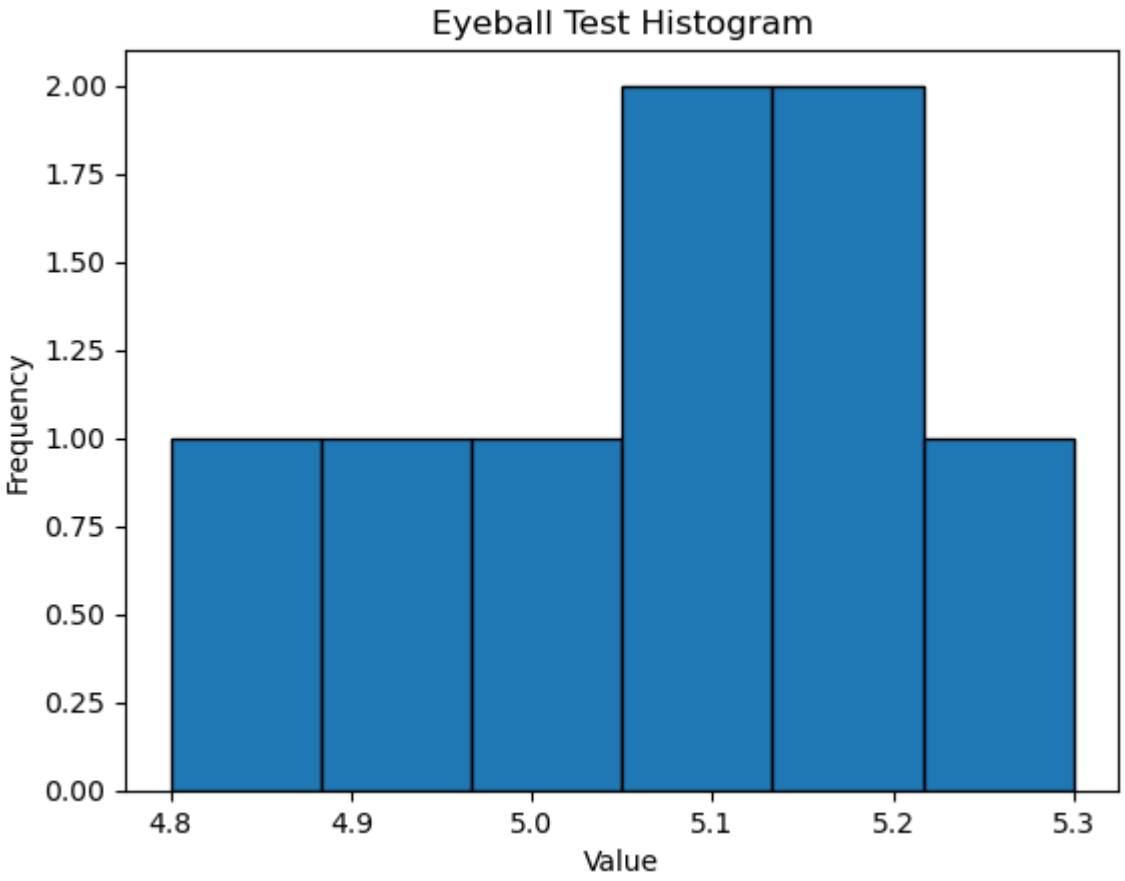
- Visual Assessment: Plotting histograms ("Eyeball test").
- Statistical Tests: Shapiro-Wilk test. (Looking to fail to reject $H_0$)

**Note**: Parametric test are typically robust to violations of this assumption, meaning the data does not need to be perfectly normal.

## Eyeball test example

In [16]:
```
import matplotlib.pyplot as plt

# Example data (replace with your own observed data)
data = [4.8, 5.2, 5.1, 4.9, 5.3, 5.0, 5.2, 5.1]

# Plot histogram
plt.hist(data, bins=6, edgecolor='black')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Eyeball Test Histogram')
```

```
plt.grid(False)
plt.show()
```




## Shapiro-Wilk example

In [17]:
```
from scipy.stats import shapiro

# Example data (replace with your own data)
data = [4.8, 5.2, 5.1, 4.9, 5.3, 5.0, 5.2, 5.1]

# Perform Shapiro-Wilk test
statistic, p_value = shapiro(data)
print("Shapiro-Wilk Test Results:")
print("Test Statistic:", round(statistic, 2))
print("p-value:", round(p_value, 2))
```

```
Shapiro-Wilk Test Results:
Test Statistic: 0.96
p-value: 0.8
```

## Independence Assumption

- Study Design: Ensure independence through random sampling or experimental design.

## Homogeneity of Variance Assumption

- Levene's test: Test for equality of variances. (Looking to reject $H_0$)

## Example using Levene's test

In [18]:
```
from scipy.stats import levene

# Example data (replace with your own data)
group1 = [15, 18, 20, 22, 17]
group2 = [12, 14, 16, 15, 18]

# Perform Levene's test
statistic, p_value = levene(group1, group2)
print("Levene's Test Results:")
print("Test Statistic:", round(statistic, 2))
print("p-value:", round(p_value, 2))
```

```
Levene's Test Results:
Test Statistic: 0.19
p-value: 0.68
```

# End of chapter questions

Using the Palmer Penguin data set, introduced in previous chapters:

1. Test whether the mean weight of Gentoo penguins is 400 g

2. Compare between the bill lengths of male Adelie and Chinstrap penguins

> Make sure to confirm that all assumptions of the test are met

In [ ]:
```

```

# Chapter 11: Analysis of Variance (ANOVA)

Learning Goals:

1. Understand the concept and applications of Analysis of Variance (ANOVA).
2. Learn about different types of ANOVA designs.
3. Explore the assumptions underlying ANOVA and how to assess them.
4. Learn how to perform ANOVA in Python and interpret the results.

# Introduction

Analysis of Variance (ANOVA) is a statistical technique used to compare the means of two or more groups or conditions. It determines whether the variability between group means is larger than the variability within groups.

## Applications of ANOVA

ANOVA is commonly used in various scenarios, including:

- Comparing means across multiple treatment groups.
- Investigating the effects of categorical and continuous variables.
- Assessing interactions between variables.

## One-Way ANOVA

The one-way ANOVA compares the means of two or more independent groups or conditions.

$H_0: \mu_1 = \mu_2 = \mu_i$

$H_A: \text{At least one} \mu_i \text{is different}$

$F = \frac{MS_{groups}}{MS_{error}}$

# Calculate F value

### Partition Sum or squares

$SS_{total} = SS_{error} + SS_{groups}$

n = sample size of $y_i$

N = total number of observations

Grand mean: $\bar{Y} = \frac{\sum{n_i}{\bar{y}_i}}{N}$

where

$N = \sum(n_i)$

$SS_{groups} = \sum{n_i(\bar{y}_i - \bar{Y})^2}$

$SS_{error} = \sum{s^{2}_{i}(n_i - 1)}$

### Calculate mean squares

k = number of groups

$MS_{groups} = \frac{SS_{groups}}{df_{groups}}$

where

$df_{groups} = k - 1$

$MS_{error} = \frac{SS_{error}}{df_{error}}$

where

$df_{error} = N - k$

## ANOVA table

| Source of variance | Sum of squares | df | Mean squares | F | P-value |
|---|---|---|---|---|---|
| Groups | $SS_{groups}$ | groups – 1 | $MS_{groups}$ | $F = MS_{groups} / MS_{error}$ | P-value |
| Error | $SS_{error}$ | observations – groups | $MS_{error}$ | | |
| Total | $SS_{total}$ | $df_{error} + df_{groups}$ | | | |

## Two-Way ANOVA

The two-way ANOVA compares the means across two categorical independent variables (factors) and their interactions.

## Assumptions of ANOVA

Assumption of Normality

- Visual Assessment: Q-Q plots, histograms.
- Statistical Tests: Shapiro-Wilk test, Anderson-Darling test.

Assumption of Homogeneity of Variances

- Visual Assessment: Box plots, Levene's test.
- Statistical Test: Levene's test.

## Performing ANOVA in Python

In [13]:
```
from scipy.stats import f_oneway

# Example data (replace with your own data)
group1 = [5, 6, 8, 7, 9]
group2 = [4, 7, 6, 9, 8]
group3 = [2, 4, 6, 5, 7]

# Perform one-way ANOVA
f_statistic, p_value = f_oneway(group1, group2, group3)
print("One-Way ANOVA Results:")
print("F-statistic:", round(f_statistic,2))
print("p-value:", round(p_value, 2))
```

```
One-Way ANOVA Results:
F-statistic: 2.24
p-value: 0.15
```

In [14]:
```
import statsmodels.api as sm
import pandas as pd
from statsmodels.formula.api import ols

# Example data (replace with your own data)
data = {'A': ['A1', 'A2', 'A1', 'A2', 'A1', 'A2', 'A1', 'A2'],
        'B': ['B1', 'B1', 'B2', 'B2', 'B1', 'B1', 'B2', 'B2'],
        'Value': [5, 7, 4, 6, 8, 10, 9, 11]}

# Create DataFrame
df = pd.DataFrame(data)

# Perform two-way ANOVA
model = ols('Value ~ A + B + A:B', data=df).fit()
anova_table = round(sm.stats.anova_lm(model), 3)
print("Two-Way ANOVA Results:")
print(anova_table)
```

```
Two-Way ANOVA Results:
            df  sum_sq  mean_sq      F  PR(>F)
A          1.0     8.0      8.0  0.941   0.387
B          1.0     0.0      0.0  0.000   1.000
A:B        1.0     0.0      0.0  0.000   1.000
Residual   4.0    34.0      8.5    NaN     NaN
```

## Interpreting ANOVA Results

One-Way ANOVA

- Evaluate the significance level (p-value).

- If the p-value is below the predetermined significance level (e.g., 0.05), reject the null hypothesis and conclude that there are significant differences between the group means.

Two-Way ANOVA

- Examine the main effects of each factor.
- Assess the interaction effect between the factors.
- Consider the significance levels (p-values) to determine the presence of significant effects.

## End of chapter question

Using the Palmer Penguin data set, compare the weights of female penguins of all three species using an ANOVA.

In [ ]:

# Chapter 12: Data transformations and non-Parametric Statistical Tests

Learning Goals:

1. Understand the concept and applications of non-parametric tests.
2. Learn about common non-parametric tests and their parametric counterparts.
3. Explore the advantages and limitations of non-parametric tests.
4. Utilize Python code examples to perform non-parametric tests.

## Introduction

Up to this point, we have been testing whether the assumptions of our particular test has been met by our data. However, it is not guaranteed that our assumptions will be met in all cases. If we find that the assumptions have been violated, we have a few options.

1. Ignore violations of assumptions: This usually only works when we are comparing means and the **normality assumption** is violated. If our *n* is high, we can ignore slight non-normality. We can't really do this for differences in variance, especially if our samples are small or our uneven.

2. Transform your data: We can use different transformations on the data and then retest the assumptions. **Important:** While analyses should be done using transformed data, data should be presented and reported untransformed, *and* all data used in a particular study needs to be transformed using the same technique. Examples:

- Natural log transformation: ln(y + 1)
- Log 10 transformation: log10(y + 1)

- Square-root transformation: sqrt(y + 0.5)
- Arcsine square root transformation (for proportions): arcsin(sqrt(p))
- Reciprocal transformation: 1 / y

3. Use nonparametric tests: non-parametric tests are statistical tests that do not rely on assumptions about the underlying distribution of the data. They are used when data do not meet the assumptions of parametric tests or when the research question involves categorical or ranked data. Nonparametric tests tend to have a **lower power** than parametric tests.

## Applications of Non-Parametric Tests

Non-parametric tests are commonly used in various scenarios, including:

- Comparing medians or distributions across groups.
- Analyzing ranked or ordinal data.
- Testing for independence or association between variables.

# Common Non-Parametric Tests

## Mann-Whitney U Test (Replacement for two sample t-test)

The Mann-Whitney U test compares medians between two independent groups.

In [4]:
```
from scipy.stats import mannwhitneyu

# Example data (replace with your own data)
group1 = [6, 7, 8, 9, 10]
group2 = [3, 5, 6, 7, 9]

# Perform Mann-Whitney U test
statistic, p_value = mannwhitneyu(group1, group2)
print("Mann-Whitney U Test Results:")
print("Test Statistic:", round(statistic, 2))
print("p-value:", round(p_value, 2))
```

```
Mann-Whitney U Test Results:
Test Statistic: 19.5
p-value: 0.17
```

## Wilcoxon Signed-Rank Test (Replacement for Paired t-test)

The Wilcoxon Signed-Rank test compares medians between paired or matched samples.

In [5]:
```
from scipy.stats import wilcoxon

# Example data
before = [5, 7, 6, 4, 9]
```

```
after = [6, 8, 7, 5, 11]

# Perform Wilcoxon Signed-Rank test
statistic, p_value = wilcoxon(before, after)
print("Wilcoxon Signed-Rank Test Results:")
print("Test Statistic:", round(statistic, 2))
print("p-value:", round(p_value, 2))
```

```
Wilcoxon Signed-Rank Test Results:
Test Statistic: 0.0
p-value: 0.06
```

### Kruskal-Wallis H Test (Replacement for One-Way ANOVA)

The Kruskal-Wallis H test compares medians across two or more independent groups.

In [6]:
```
from scipy.stats import kruskal

# Example data (replace with your own data)
group1 = [5, 6, 7, 8, 9]
group2 = [3, 5, 6, 7, 9]
group3 = [2, 4, 6, 5, 7]

# Perform Kruskal-Wallis H test
statistic, p_value = kruskal(group1, group2, group3)
print("Kruskal-Wallis H Test Results:")
print("Test Statistic:", round(statistic, 2))
print("p-value:", round(p_value, 2))
```

```
Kruskal-Wallis H Test Results:
Test Statistic: 2.83
p-value: 0.24
```

## Advantages and Limitations of Non-Parametric Tests

Advantages

- Non-reliance on distributional assumptions.
- Robustness against outliers.

Limitations

- Less statistical power compared to parametric tests.
- Restricted to certain types of research questions.

## End of chapter question

Using the Palmer Penguin data set, compare the weights of male penguins of all three species using the non-parametric alternative to the ANOVA.

# Chapter 13: Predictive Models, Correlation and Regression Analysis

Learning Goals:

1. Understand the concepts of correlation and regression analysis.
2. Learn how to measure and interpret correlation between variables.
3. Explore simple linear regression and its applications.
4. Utilize Python code examples to perform correlation and regression analysis.

## Introduction

Correlation measures the strength and direction of the relationship between two variables. It helps determine whether variables are positively, negatively, or not correlated. Regression analysis models the relationship between a dependent variable and one or more independent variables. It helps predict and understand the impact of independent variables on the dependent variable.

## Types of Correlation

- Pearson correlation: Measures linear correlation between continuous variables.
- Spearman correlation: Measures monotonic (non-linear) correlation or correlation between ranked variables.

## Measuring and Interpreting Correlation

The **Pearson correlation coefficient**, denoted as r, ranges from -1 to 1. It indicates the strength and direction of the linear relationship between variables.

The **Spearman correlation coefficient**, denoted as ρ (rho), also ranges from -1 to 1. It assesses the monotonic relationship between variables.

In [1]:
```
import numpy as np
import scipy.stats as stats

# Example data (replace with your own data)
x = [1, 2, 3, 4, 5]
y = [5, 7, 8, 9, 11]

# Calculate Pearson correlation coefficient
pearson_corr, _ = stats.pearsonr(x, y)
print("Pearson correlation coefficient:", round(pearson_corr, 2))

# Calculate Spearman correlation coefficient
```

```
spearman_corr, _ = stats.spearmanr(x, y)
print("Spearman correlation coefficient:", round(spearman_corr, 2))
```

```
Pearson correlation coefficient: 0.99
Spearman correlation coefficient: 1.0
```

# Correlation Interpretation

Pearson correlation coefficient (r):

- r > 0: Positive correlation.
- r < 0: Negative correlation.
- r ≈ 0: Weak or no correlation.

Spearman correlation coefficient (ρ):

- ρ > 0: Monotonic positive correlation.
- ρ < 0: Monotonic negative correlation.
- ρ ≈ 0: Weak or no monotonic correlation.

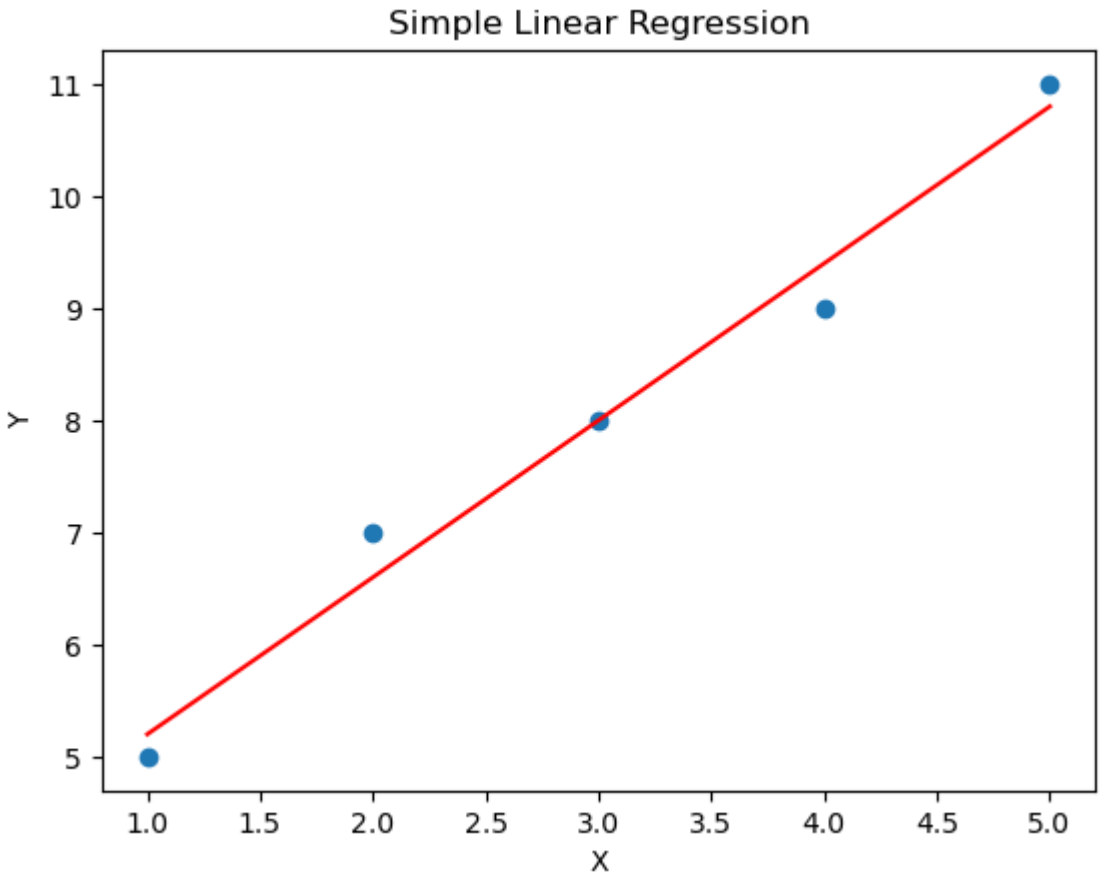
# Simple Linear Regression

Simple linear regression models the relationship between two continuous variables using a straight line equation

$y = mx + b$

In [2]:
```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

# Example data (replace with your own data)
x = [1, 2, 3, 4, 5]
y = [5, 7, 8, 9, 11]

# Perform simple linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)

# Plot the regression line
plt.scatter(x, y)
plt.plot(x, slope * np.array(x) + intercept, color='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Simple Linear Regression')
plt.show()

print("Slope:", round(slope, 3))
print("Intercept:", round(intercept, 3))
print("R-value (Correlation Coefficient):", round(r_value, 3))
```

```
print("P-value:", round(p_value, 3))
print("Standard Error:", round(std_err, 3))
```




```
Slope: 1.4
Intercept: 3.8
R-value (Correlation Coefficient): 0.99
P-value: 0.001
Standard Error: 0.115
```

## Regression Interpretation

- Slope: Represents the change in the dependent variable per unit change in the independent variable.
- Intercept: Represents the predicted value of the dependent variable when the independent variable is zero.
- R-value: Indicates the strength and direction of the linear relationship between variables.
- P-value: Determines the statistical significance of the relationship.
- Standard Error: Provides a measure of the accuracy of the regression estimates.

## End of chapter question

Using the Palmer Penguin data, determine how well female Gentoo penguin weight and flipper length correlate and how well the change in weight predicts flipper length. Make sure to provide both numeric and graphical outputs.

In [ ]:

# Chapter 14: Machine Learning and AI

## Introduction

Machine learning is a subfield of artificial intelligence (AI) that involves the development of algorithms and models that can learn from and make predictions or decisions based on data. There are many different types of machine learning algorithms, each with their own strengths and weaknesses, and they can be applied to a wide variety of problems.

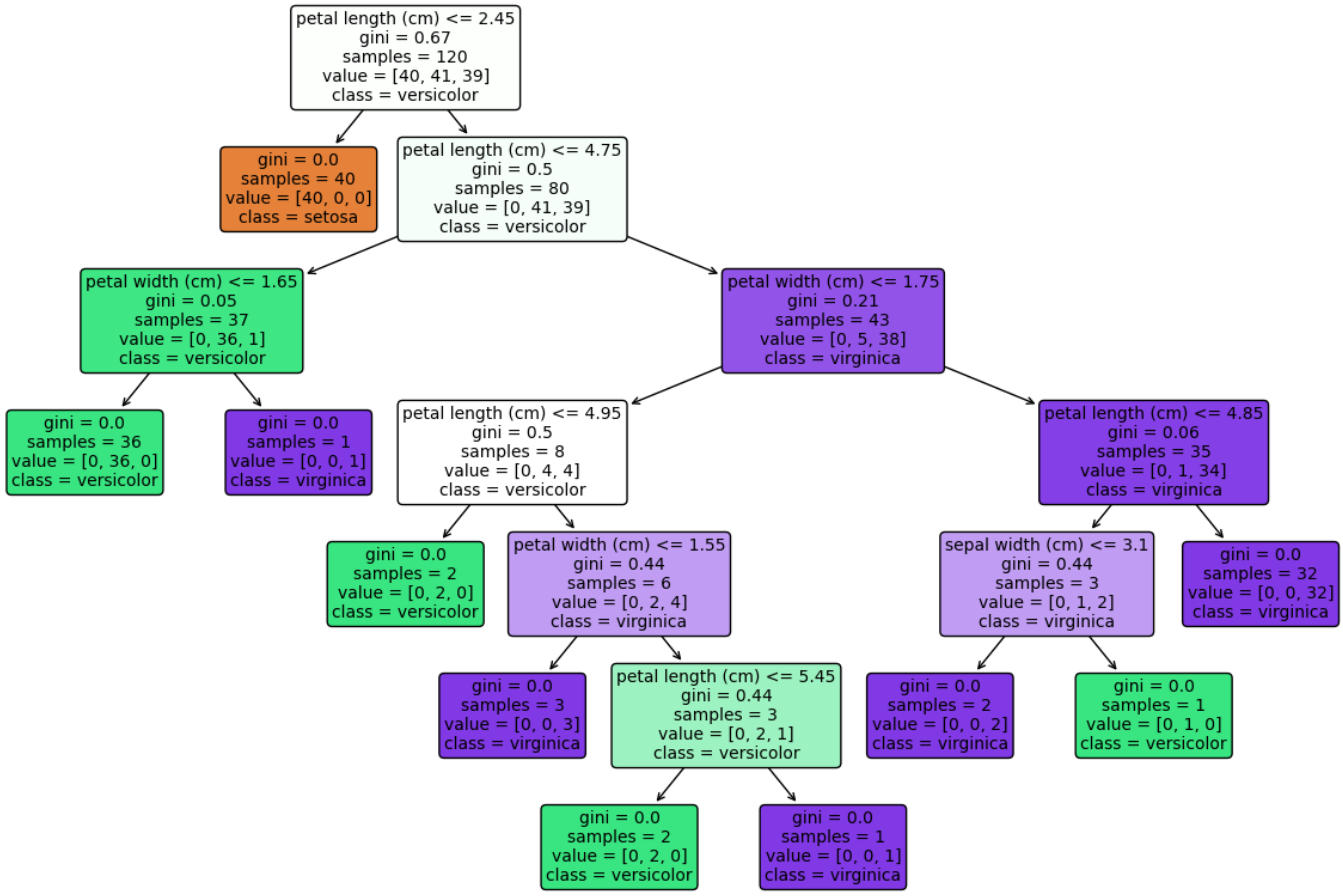
## Types of machine learning models

### Decision trees

One of the most popular and widely used types of machine learning algorithms is decision trees. Decision trees are a type of algorithm that can be used for both classification and regression problems. They work by recursively partitioning the data into subsets based on the values of the input features, and making predictions based on the majority class or mean value of the observations in each subset.

In [8]:
```python
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load data
iris = load_iris()
X = iris.data
y = iris.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build decision tree
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model's performance
from sklearn.metrics import accuracy_score
print("Accuracy:", accuracy_score(y_test, y_pred))

# Plot the decision tree with enhanced visualization
plt.figure(figsize=(15, 10))
plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names,
          filled=True, rounded=True, fontsize=10, precision=2, impurity=True)
plt.show()
```

```
Accuracy: 1.0
```




In this example, we are using the **iris dataset**, that contains measurements of sepals and petals of three different species of irises. We are using all the columns as predictors and the species as response.

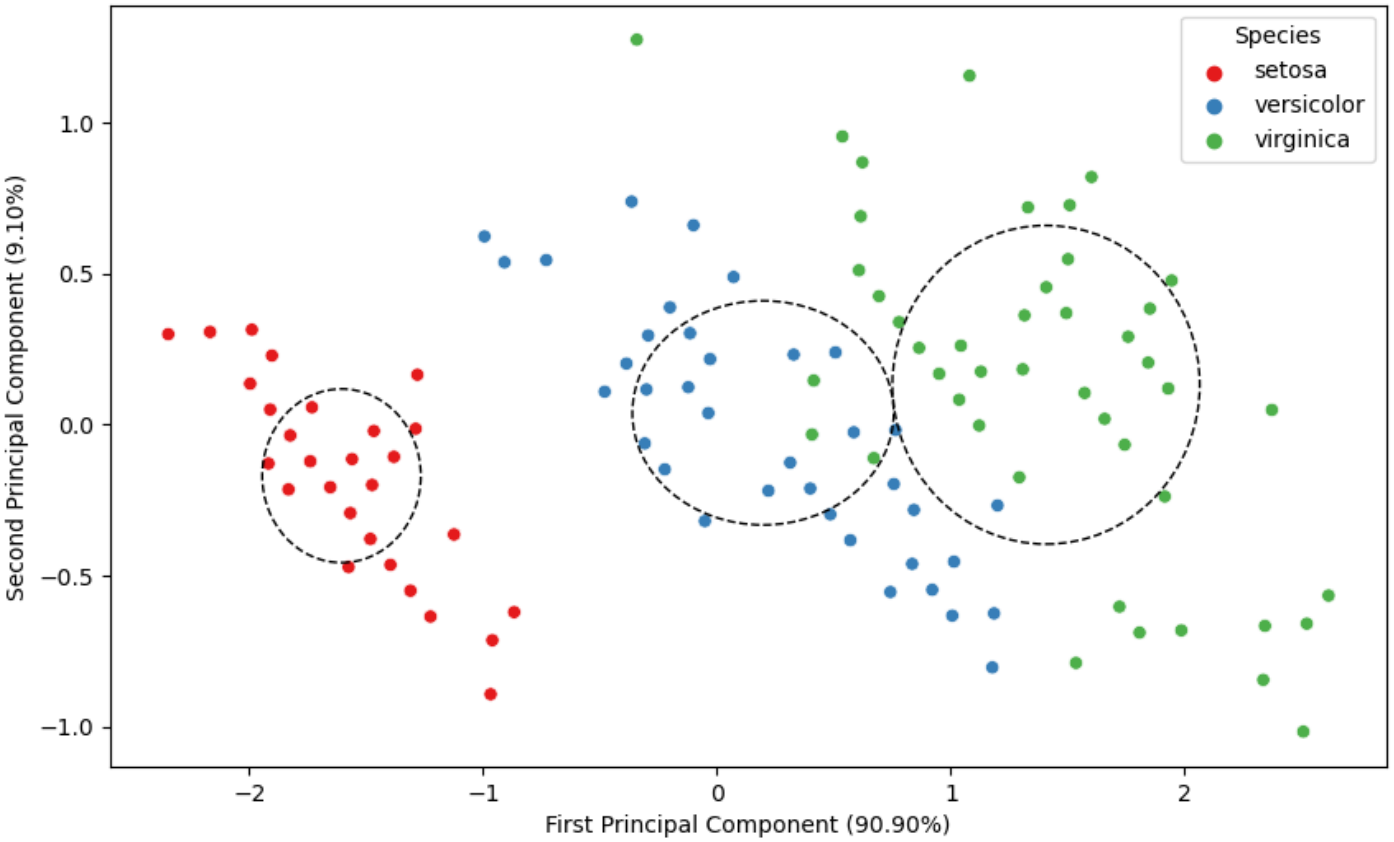
The `train_test_split()` function is used to split the data into a training set and a test set, the `DecisionTreeClassifier()` function is used to build the decision tree model and the fit function is used to train the model. The predict function is used to make predictions on the test set, and the `accuracy_score()` function is used to evaluate the model's performance by comparing the predicted values to the actual values.

We also plot the tree using `plot_tree()` to visualize the tree. We can see that petal length less than or equal to 2.45 cm is confidently identified as *Iris setosa*. We can also see that *Iris versicolor* and *Iris virginica* identification takes more steps. This is likely because the physical features of these two species are more similar than to those of *I. setosa*.

We can explore this further by using Principal Component Analysis (PCA), which is a form of summary statistic, not machine learning. It does this by finding a new set of uncorrelated variables, called **principal components**, which can be used to represent the original data.

In [9]:
```
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import Ellipse
```

```python
# Import data
url = "https://raw.githubusercontent.com/lundquist-ecology-lab/biostatistics/main/example_data/ir
data = pd.read_csv(url)

# Scale data using the same method as in R
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data.iloc[:, [0, 3]])

# Perform PCA with svd_solver='auto'
pca = PCA(n_components=2, svd_solver='auto')
pca.fit(data_scaled)
data_pca = pca.transform(data_scaled)

# Plot PCA with ellipses around groups
plt.figure(figsize=(10, 6))

# Create a scatter plot
scatter = sns.scatterplot(x=data_pca[:, 0], y=data_pca[:, 1], hue=data['Species'], palette='Set1

# Add ellipses around groups
for spec in data['Species'].unique():
    ell_data = data_pca[data['Species'] == spec]
    ellipse = Ellipse(
        (ell_data[:, 0].mean(), ell_data[:, 1].mean()),
        width=2 * ell_data[:, 0].std(),
        height=2 * ell_data[:, 1].std(),
        fill=False,
        color='black',
        linestyle='dashed'
    )
    scatter.add_patch(ellipse)

# Add percentage of variance explained on axes
explained_var = pca.explained_variance_ratio_
plt.xlabel(f"First Principal Component ({explained_var[0] * 100:.2f}%)")
plt.ylabel(f"Second Principal Component ({explained_var[1] * 100:.2f}%)")

# Add legend
plt.legend(title='Species')

# Show the plot
plt.show()
```




You can see that the first principal component (PC1) explains 90.9% of the variance in the data. The next principal component (PC2) explains an additional 9.1%. The ellipses represent potential groupings of data based on the principal components. In the example above, we can see that the *Iris* species form three relatively clear clusters (particularly *I. setosa*). It makes sense that it would be more complicated to for the machine learning model to decern between *I. versicolor* and *I. virginica* based on our data.

## Linear models

Another popular type of machine learning algorithm is linear regression. Linear regression is a type of algorithm that can be used for continuous target variables and it attempts to find the best linear relationship between the input features and the target variable.

In [12]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Load data
url = 'https://gist.githubusercontent.com/ZeccaLehn/4e06d2575eb9589dbe8c365d61cb056c/raw/898a40b0
data = pd.read_csv(url, index_col=0)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[['wt','hp']], data['mpg'], test_size=0.

# Build linear regression model
reg = LinearRegression()
reg.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = reg.predict(X_test)

# Evaluate the model's performance
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

```
8.474832262021078
```

In this example, we are using the **mtcars** dataset, a dataset that contains measurements of different cars such as the miles per gallon, weight, and horsepower. We are using the weight and horsepower as predictors and the miles per gallon as the target variable.

The predictors are the weight and horsepower and the target variable is the miles per gallon. The `train_test_split()` function is used to split the data into a training set and a test set, the `LinearRegression()` function is used to build the linear regression model, and the fit function is used to train the model. The predict function is used to make predictions on the test set, and the `mean_squared_error()` function is used to evaluate the model's performance by comparing the predicted values to the actual values. The lower the mean squared error (MSE), the better.

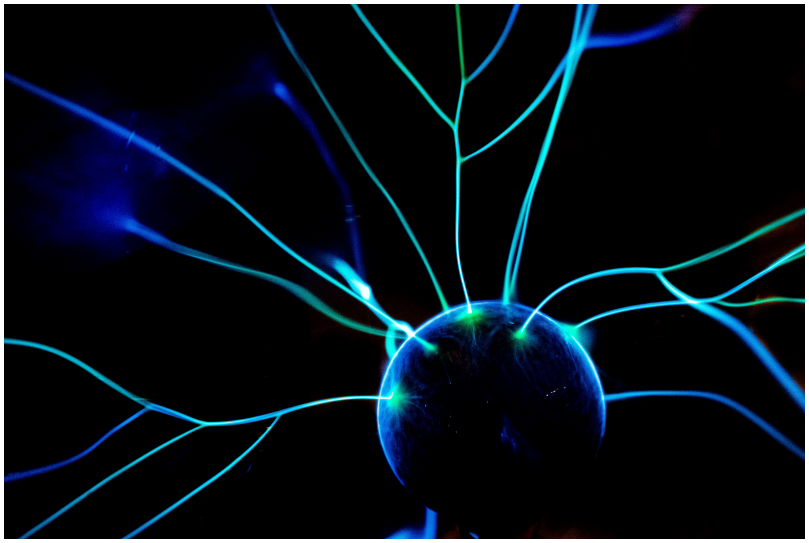
Below, we take the same model and only use horsepower as the predictor. Notice that the MSE is much higher, indicating that this model is less predictive than the previous one taking into account both weight and horsepower.

In [13]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Load data
url = 'https://gist.githubusercontent.com/ZeccaLehn/4e06d2575eb9589dbe8c365d61cb056c/raw/898a40b0
data = pd.read_csv(url, index_col=0)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[['hp']], data['mpg'], test_size=0.2, ra

# Build linear regression model
reg = LinearRegression()
reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = reg.predict(X_test)

# Evaluate the model's performance
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

```
24.006401948028003
```

## Artificial neural networks (ANNs)

Finally, another popular type of machine learning algorithm is artificial neural networks (ANNs). Artificial neural networks are a type of algorithm that are inspired by the structure and function of the human brain. They are composed of layers of interconnected "neurons" that process and transmit information. They are highly flexible and can be used for a wide variety of problems, including image and speech recognition, natural language processing, and more.



In [14]:
```python
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load data
iris = load_iris()
X = iris.data
y = iris.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build ANN model
clf = MLPClassifier(hidden_layer_sizes=(10,),max_iter=1000)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model's performance
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
1.0
```

```
/home/mlundquist/.local/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.
  warnings.warn(
```

This example again uses the iris dataset. We are using all the columns as predictors and the species as response. The `train_test_split()` function is used to split the data into a training set and a test set, the

`MLPClassifier()` function is used to build the ANN model, the fit function is used to train the model, the predict function is used to make predictions on the test set, and the `accuracy_score()` function is used to evaluate the model's performance by comparing the predicted values to the actual values.

# Generative AI (as written by ChatGPT)

ChatGPT (a generative AI) is a state-of-the-art language model developed by OpenAI. It is based on the transformer architecture, which has been shown to be highly effective for a variety of natural language processing tasks. The model is trained on a massive dataset of text from the internet, allowing it to generate human-like responses to a wide range of prompts.

## ChatGPT for biostatistics students

ChatGPT can potentially be used by biostatistics students to check their R or Python code for their homework, as it can generate explanations of the results and outputs of the code. However, it's important to note that ChatGPT is not a replacement for human understanding and expertise, and students should still strive to understand the concepts and methods behind the code they are writing. Additionally, the model's understanding of the code and the results is limited by the complexity of the underlying models and the quality of the input data.

It's also important to note that using a model like ChatGPT to check homework could be considered as plagiarism in some cases, as it could be considered as the work of another person, so it's important to check with your instructor regarding the use of such tools before using it.

It's also possible to use ChatGPT to generate some examples, but students should be aware that the model is not aware of the context of the homework and it could generate examples that are not appropriate for the homework.

In short, ChatGPT can be a useful tool for biostatistics students to check their code, but it should be used with caution and with the understanding that it is not a replacement for human expertise and

understanding.

### ChatGPT in professional biostatistics

One potential application of ChatGPT in biostatistics is in the automated generation of research reports and manuscripts. For example, a researcher could provide the model with a dataset and a set of analysis commands in R or Python, and the model could generate a report detailing the results of the analysis. This could save researchers a significant amount of time and effort, as they would not need to manually write up the results of their analyses.

However, it's important to note that ChatGPT is not a replacement for human expertise and understanding. The model is only as good as the data it was trained on, and it may make mistakes or miss important details if the input data is noisy or incomplete. Additionally, the model's ability to understand and interpret the results of statistical analyses is limited by the complexity of the underlying models and the quality of the input data.

An example of how ChatGPT can be used in Python to generate a report on a logistic regression analysis:

In [16]:
```
%%script false --no-raise-error
# Load data
import pandas as pd
data = pd.read_csv("data.csv")

# Run logistic regression analysis
from sklearn.linear_model import LogisticRegression
X = data[['age', 'income']]
y = data['default']
clf = LogisticRegression(random_state=0).fit(X, y)

# Provide ChatGPT with the summary of the model
summary_model = clf.coef_

# ChatGPT generates a report
ChatGPT_output = "The logistic regression model shows that the age and income variables are both
```

**NOTE**: The data used in the Python example is a made up data set and is not based on any real-world dataset. It's just an example to illustrate how ChatGPT can be used to generate a report on a logistic regression analysis. In a real-world scenario, the researcher would need to provide the model with a dataset in a format that can be read by the programming language being used. For example, a .csv file for Python.

## Conclusion

Machine learning is a powerful tool that can be used to extract insights from data and make predictions and decisions. There are many different types of machine learning algorithms, each with their own strengths and weaknesses, including decision trees, linear regression, and artificial neural networks. These examples demonstrate how to implement these algorithms, but it's important to note that the performance of these models may vary depending on the specific problem and the quality of the data. It's also important to understand the underlying concepts of each algorithm and the assumptions they make, to make the right

choice for a given problem. In other words, there is no single best test, it is up to the biostatistician to decide on the best technique to solve their particular problem.

# Appendix 1: Python and Jupyter basics

## Introduction to Python

Python is a high-level, interpreted programming language known for its readability and versatility. With its vast ecosystem of scientific computing libraries such as NumPy, SciPy, pandas, and matplotlib, Python is a popular choice for data analysis in diverse fields, including biostatistics.

In biostatistics, Python is utilized for data wrangling, statistical analysis, machine learning, and visualization, among other tasks. Whether you're processing large genomic datasets, modeling biological systems, or testing hypotheses with advanced statistical methods, Python offers a suite of tools to efficiently carry out these tasks.

## Python Syntax Basics

Let's take a look at some fundamental Python syntax that you'll use frequently for biostatistics:

### Variables and Types:

Python has several basic types like integers, floats, booleans, and strings. You can assign values to variables without declaring their type:

In [5]:
```
x = 10  # Integer
y = 3.14  # Float
s = "Biostatistics"  # String
b = True  # Boolean
```

### Lists and Dictionaries

Python includes several compound types, two of which are lists and dictionaries.

In [6]:
```
# A list is an ordered collection of values
samples = ["sample1", "sample2", "sample3"]

# A dictionary is an unordered collection of key-value pairs
sample_dict = {"sample1": 23, "sample2": 45, "sample3": 37}
```

### Control Flow

Python uses control flow structures like conditionals and loops:

In [7]:
```python
# If-Else Conditional
if x > 5:
    print("x is greater than 5")
else:
    print("x is less than or equal to 5")

# For Loop
for sample in samples:
    print(sample)
```

```
x is greater than 5
sample1
sample2
sample3
```

### Functions

Functions are reusable blocks of code. Python has numerous built-in functions and allows you to define your own:

In [ ]:
```python
# Defining a function
def square(x):
    return x ** 2

# Using a function
y = square(5)  # y is now 25
```

# Introduction to Jupyter Notebook

Jupyter Notebook is an open-source web application that enables you to create and share documents containing live Python code, visualizations, and narrative text.

Python.GriffyStats.com is an online platform that hosts Jupyter Notebooks, requiring only a login without any installation on your personal computer.

## Accessing Jupyter Notebook

**Note: You will need to sign up for a free GitHub.com account.**

Navigate to python.griffystats.com in your web browser and log in with your GitHub credentials. After logging in, you'll access the Jupyter Notebook dashboard, where you can create or open notebooks.

## Jupyter Notebook Interface

A Jupyter Notebook consists of a series of cells, which can be categorized into:

1. **Code Cells**: You can write and execute Python code here.
2. **Markdown Cells**: These cells allow you to write explanatory text, equations, or add images using Markdown, a lightweight markup language.

### Working with Cells

In a **Code Cell**, you can write Python code and execute it by hitting Shift+Enter:

In [8]:
```
# Python code
print("Welcome to Biostatistics with Python!")
```

```
Welcome to Biostatistics with Python!
```

A **Markdown Cell** lets you write text, insert images, or write equations in LaTeX. Here's a simple example:

# Welcome to Biostatistics

This is a course on **Biostatistics** with *Python*.

In [ ]:
```

```

# Appendix 2: Working with Jupyter Notebook

## 1: Open Jupyter Notebook

Launch `python.griffystats.com` in your web browser.

Navigate to the desired directory (e.g. "lessons") where your notebook is located.

## 2: Open the Notebook

Click on the notebook file with the extension `.ipynb` to open it.

## 3: Save the Notebook

Make any desired changes to the notebook.

To save the notebook, click on the "Save" button in the toolbar or press `Ctrl + S` (Windows/Linux) or `Command + S` (Mac).

## 4: Export the Notebook

There are various options to export your notebook, depending on the desired format.

Exporting as HTML:

- Go to the "File" menu.

- Select "Download as".
- Choose "HTML (.html)".

Exporting as PDF:

- Install the necessary dependencies for exporting as PDF, if not already installed. You may need `pandoc` and `TeX` distribution like `MiKTeX` or `TeX Live`.
- Go to the "File" menu.
- Select "Download as".
- Choose "PDF via LaTeX (.pdf)".

Exporting as Markdown:

- Go to the "File" menu.
- Select "Download as".
- Choose "Markdown (.md)".

Exporting as Python script:

- Go to the "File" menu.
- Select "Download as".
- Choose "Python (.py)".

Exporting as other formats:

- Jupyter Notebook also supports exporting notebooks to formats such as LaTeX, reStructuredText, executable script, etc. You can explore these options in the "Download as" menu.

## 5: Verify and Save the Exported File

- After selecting the desired export format, the notebook will be downloaded to your local machine.
- Verify that the exported file is saved in the designated location on your computer.

In [ ]:

# Appendix 3: Importing and Modifying Data for biostatistics

Learning goals

1. Import data of different formats into Python using Pandas.
2. Identify and handle missing values in a dataset.
3. Transform and reshape data to suit the needs of your analysis.
4. Understand how to merge multiple datasets into a single, coherent dataset.

# Introduction

In the realm of biostatistics, dealing with data is a key part of the process. Often, this data is not collected with the intention of being processed in Python, leading to potential challenges. This chapter provides an overview of importing and modifying data in Python from a biostatistics perspective. We will discuss common methods for importing data, techniques for handling missing values, and ways to transform and reshape your data to suit your analysis.

Before we begin, ensure you have the following essential Python libraries installed:

- Pandas: Provides data structures and data analysis tools.
- Numpy: Allows numerical operations on arrays and matrices.
- Matplotlib: Facilitates the creation of static, animated, and interactive visualizations in Python.

You can install these packages using pip:

In [17]:
```
%%script false --no-raise-error

pip install pandas numpy matplotlib
```

## Importing Data

Data in biostatistics can come in various formats such as CSV (Comma-Separated Values), Excel (xls, xlsx), or TSV (Tab-Separated Values). We'll primarily use the Pandas library to read these files into Python:

In [18]:
```
%%script false --no-raise-error

import pandas as pd

# Loading a CSV file
df_csv = pd.read_csv('filename.csv')

# Loading an Excel file
df_excel = pd.read_excel('filename.xlsx')

# Loading a TSV file
df_tsv = pd.read_csv('filename.tsv', sep='\t')
```

## Dealing with Missing Values

Real-world datasets often have missing values, and these missing values can lead to misleading results if not handled properly. Let's look at some techniques to handle missing values:

In [19]:
```
%%script false --no-raise-error

import pandas as pd

# Check for missing values
print(df_csv.isnull().sum())

# Drop rows with missing values
df_no_missing = df_csv.dropna()

# Fill missing values with a specified value (e.g., mean, median, mode, or a constant value)
df_fill_mean = df_csv.fillna(df_csv.mean())
```

## Data Transformation and Reshaping

Often, data may not be in a form that is ready for analysis, and you may need to reshape or transform it:

In [4]:
```
%%script false --no-raise-error

import pandas as pd

# Renaming columns
df_csv.rename(columns={'OldName1': 'NewName1', 'OldName2': 'NewName2'}, inplace=True)

# Converting a continuous variable into a categorical variable
bins = [0, 10, 20, 30, 40]
labels = ['0-10', '10-20', '20-30', '30-40']
df_csv['AgeGroup'] = pd.cut(df_csv['Age'], bins=bins, labels=labels)

# Pivoting data (reshaping)
df_pivot = df_csv.pivot(index='ColumnNameToIndex', columns='ColumnNameToColumns', values='ColumnN

# Merging two dataframes
df_merge = pd.merge(df_csv1, df_csv2, on='CommonColumnName', how='inner') # Can be 'outer', 'left
```

# Conclusion

The importance of being able to import, modify, and clean data cannot be understated in biostatistics. It's crucial for researchers and data scientists to familiarize themselves with these techniques to prepare their data correctly for downstream statistical analysis.

# Appendix 4: Importing and Using Libraries in Python for Biostatistics

Python, by itself, is a powerful programming language. However, it's the extensive collection of external libraries or packages that make Python a formidable tool for biostatistics. These libraries provide pre-written functionality to carry out numerous tasks, saving you from having to code these functionalities from scratch.

## Installing Libraries

Python libraries can be installed using a package manager like pip. For instance, to install a library like pandas, you would use the following command in your terminal:

In [8]:
```
%%script false --no-raise-error

pip install pandas
```

Most of the time, you will need to install a bunch of libraries for a particular program (thought you only need to install them on your system once). Many Python programmers will provide a "requirements.txt" file that has all the required libraries listed. You can install all the required libraries all at once by running this command in the **Terminal**:

In [9]:
```
%%script false --no-raise-error

python3 -m pip install -r requirements.txt
```

## Importing Libraries

Once a library is installed, you need to import it to use its functionalities in your Python script. Here is how you can import some commonly used libraries in biostatistics:

In [10]:
```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
```

In these lines, `as` is used to assign a short alias to the libraries, allowing you to use this shorthand when calling functions from these libraries.

## Using Libraries

Now, let's briefly look at how these libraries could be useful in a biostatistics context:

1. **NumPy (Numerical Python)**: Provides support for arrays, matrices, and high-level mathematical functions to operate on these structures.

In [11]:
```
# Creating a numpy array
data = np.array([1, 2, 3, 4, 5])

# Calculating mean
```

```
mean = np.mean(data)
print(mean)  # Output: 3.0
```

```
3.0
```

2. **Pandas**: Offers data structures and operations for manipulating numerical tables and time series, making it perfect for data cleaning and analysis.

In [12]:
```
# Creating a pandas DataFrame
data = {'Name': ['Tom', 'Nick', 'John'], 'Age': [20, 21, 19]}
df = pd.DataFrame(data)

# Displaying the DataFrame
print(df)
```

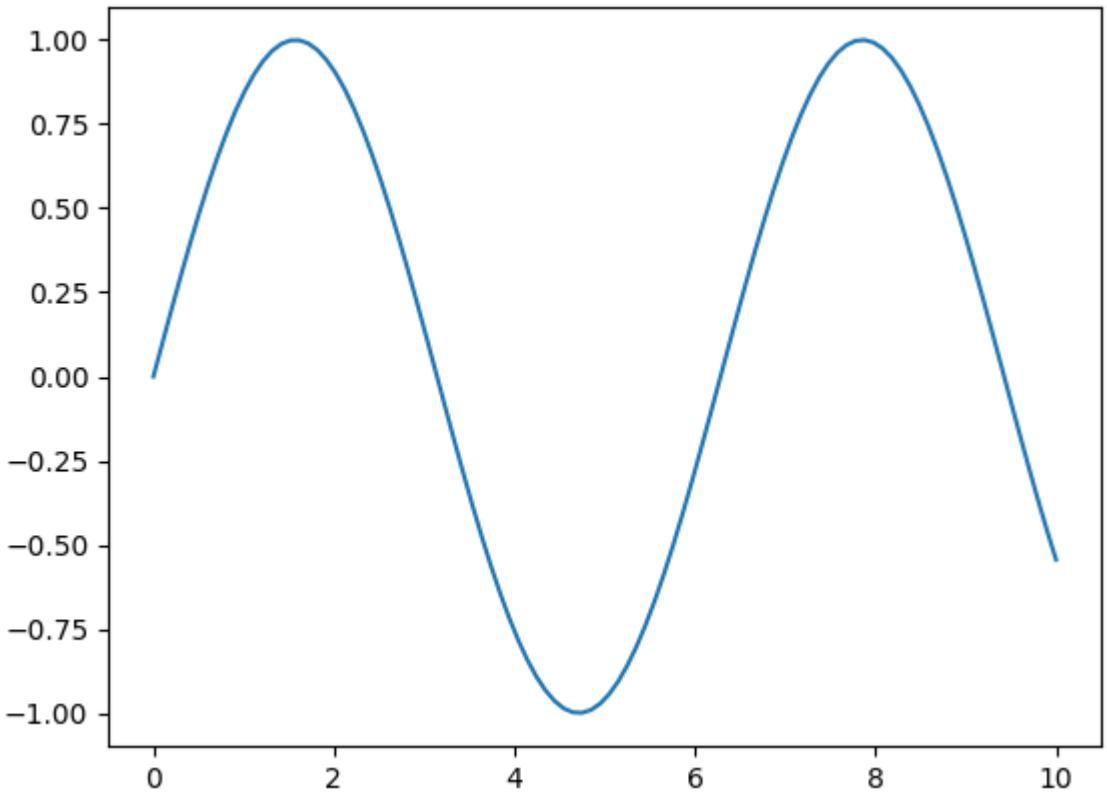
```
   Name  Age
0   Tom   20
1  Nick   21
2  John   19
```

3. **SciPy (Scientific Python)**: Builds on NumPy and provides a large number of functions that operate on numpy arrays and are useful for different types of scientific and engineering applications. The `stats` submodule provides statistical functions.

In [13]:
```
# Conducting a T-test for independent samples
group1 = np.random.normal(5, 1, 30)
group2 = np.random.normal(6, 1, 30)
t_stat, p_val = stats.ttest_ind(group1, group2)
```

4. **Matplotlib**: Provides a MATLAB-like interface for making plots and graphs.

In [14]:
```
# Creating a simple line plot
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```




In [ ]:

# Appendix 5: Jupyter Notebook tricks

## Dealing with misbehaving or non-working code

You can add the following code to the beginning of your Python code block to keep the block from being interpreted. You will see this in code blocks in chapters/lessons to example code that does not work without modification.

In [ ]:
```
%%script false --no-raise-error
```