STATISTICS NOTES

**1. What is Statistics and What Are Its Types?**

**Statistics is the branch of mathematics that deals with collecting, analyzing, interpreting, presenting, and organizing data. It enables researchers to make sense of complex data sets and draw meaningful conclusions. The two primary types of statistics are descriptive statistics, which summarize and describe the characteristics of a dataset, and inferential statistics, which use sample data to make generalizations about a larger population.**

**Types of Statistics:**

* **Descriptive Statistics:** Summarizes and organizes data using measures like mean, median, mode, and graphs.
* **Inferential Statistics:** Makes predictions or inferences about a population based on sample data.

**Example in Python:**

import numpy as np

# Descriptive Statistics Example

data = [12, 15, 14, 10, 18, 20]

mean = np.mean(data)

median = np.median(data)

mode = max(set(data), key=data.count) # Finding mode

print("Mean:", mean)

print("Median:", median)

print("Mode:", mode)

**2. Probability Introduction**

**Probability is a mathematical framework for quantifying uncertainty, measuring how likely an event is to occur. It ranges from 0 (impossible event) to 1 (certain event). Understanding probability is essential for making predictions and informed decisions based on data. It forms the foundation for inferential statistics, where conclusions are drawn from sample data.**

**Key Terms:**

* **Experiment:** A process with uncertain outcomes (e.g., rolling a die).
* **Outcome:** A possible result of an experiment.
* **Sample Space (S):** Set of all possible outcomes.

**Example in Python:**

import random

# Simulating a die roll

die\_roll = random.randint(1, 6)

print("Die Roll Outcome:", die\_roll)

**3. Addition Rule in Probability**

**Probability is a mathematical framework for quantifying uncertainty, measuring how likely an event is to occur. It ranges from 0 (impossible event) to 1 (certain event). Understanding probability is essential for making predictions and informed decisions based on data. It forms the foundation for inferential statistics, where conclusions are drawn from sample data.**

* For **mutually exclusive events** (cannot happen together):
* For **non-mutually exclusive events** (can overlap):

**Example in Python:**

# Non-mutually exclusive example

P\_A = 0.5 # Probability of A

P\_B = 0.6 # Probability of B

P\_A\_and\_B = 0.3 # Probability of A and B occurring together

P\_A\_or\_B = P\_A + P\_B - P\_A\_and\_B

print("P(A or B):", P\_A\_or\_B)

**4. Multiplication Rule in Probability**

**The multiplication rule in probability applies to independent events and states that the probability of both events A and B occurring is the product of their individual probabilities. This rule is crucial when determining the likelihood of multiple independent outcomes happening together, such as flipping a coin and rolling a die simultaneously.**

* For **independent events** (one does not affect the other):
* For **dependent events**:

**Example in Python:**

# Independent event example

P\_A = 0.4

P\_B = 0.5

P\_A\_and\_B = P\_A \* P\_B

print("P(A and B):", P\_A\_and\_B)

**5. Descriptive and Inferential Statistics**

**Descriptive statistics provide simple summaries about the sample and measures such as mean, median, mode, variance, and standard deviation. They help in understanding the basic features of the data without making any assumptions about the larger population. On the other hand, inferential statistics involve using sample data to make predictions or inferences about a population's characteristics, often involving hypothesis testing or confidence intervals.Example in Python:**

from scipy import stats

# Inferential Statistics Example

sample = [20, 21, 22, 19, 18, 20, 21]

confidence\_interval = stats.t.interval(alpha=0.95, df=len(sample)-1, loc=np.mean(sample), scale=stats.sem(sample))

print("95% Confidence Interval:", confidence\_interval)

**6. Population and Sample**

* **In statistics, a population refers to the entire group of individuals or instances about whom we seek to learn, while a sample is a subset of that population selected for analysis. The goal of sampling is to gather insights about the population without needing to collect data from every member, which can be impractical or impossible.**

**Example in Python:**

# Population vs Sample Example

population = [i for i in range(1, 101)] # Numbers 1 to 100

sample = random.sample(population, 10) # Random sample of 10 elements

print("Sample:", sample)

**7. Measure of Central Tendency (Mean, Median, Mode)**

**Measures of central tendency summarize a dataset by identifying its center point:**

* **Mean: The average value calculated by summing all values and dividing by their count.**
* **Median: The middle value when data points are arranged in ascending order.**
* **Mode: The value that appears most frequently in a dataset.  
  These measures provide insight into typical values within a dataset.**

**Example in Python:**

# Measure of Central Tendency Example

print("Mean:", np.mean(data))

print("Median:", np.median(data))

print("Mode:", mode)

**8. Measure of Dispersion (Variance, Standard Deviation)**

**Measures of dispersion describe how spread out the values in a dataset are:**

* **Variance quantifies how much the values deviate from the mean on average.**
* **Standard deviation is the square root of variance and indicates how much individual data points differ from the mean.  
  Both measures are essential for understanding variability within data and assessing consistency.**

**Example in Python:**

# Measure of Dispersion Example

variance = np.var(data)

std\_dev = np.std(data)

print("Variance:", variance)

print("Standard Deviation:", std\_dev)

**9. Population Mean and Sample Mean**

**The population mean is calculated using all members of a population and provides an exact average. In contrast, the sample mean is derived from a subset of that population and serves as an estimate for the population mean. Understanding both means is critical for inferential statistics as it helps in estimating population parameters based on sample statistics.Example in Python:**

# Population vs Sample Mean Example

print("Population Mean:", np.mean(population))

print("Sample Mean:", np.mean(sample))

**10. Sampling Methods and Their Types**

**Sampling methods refer to techniques used to select individuals from a population for analysis. Common types include:**

* **Simple Random Sampling: Every member has an equal chance of being selected.**
* **Stratified Sampling: The population is divided into strata; samples are taken from each stratum.**
* **Systematic Sampling: Selecting every nth member from a list.  
  These methods help ensure that samples are representative of the population.**

**Example in Python:**

# Random Sampling Example

random\_sample = random.sample(population, 5)

print("Random Sample:", random\_sample)

**11. Variables and Their Types**

**Variables are characteristics or properties that can take on different values across observations. They can be classified as:**

* **Qualitative Variables: Non-numeric categories (e.g., gender).**
* **Quantitative Variables: Numeric values that can be further divided into discrete (countable) or continuous (measurable) types.  
  Understanding variables is essential for statistical analysis as they dictate how data can be interpreted.**

**Example in Python:**

# Categorical vs Numeric Example

categorical = ["Red", "Blue", "Green"]

numeric = [5, 10, 15]

print("Categorical Data:", categorical)

print("Numeric Data:", numeric)

**12. Variable Measurement Scales**

**Variables can be measured on different scales:**

* **Nominal Scale: Categories without any order (e.g., colors).**
* **Ordinal Scale: Categories with a defined order (e.g., rankings).**
* **Interval Scale: Numeric scales without a true zero point (e.g., temperature).**
* **Ratio Scale: Numeric scales with a true zero point (e.g., weight).  
  These scales determine how data can be analyzed and interpreted.**

**Example:** Temperature (Celsius) is interval, Weight is ratio.

**13. Frequency Distribution and Cumulative Frequency**

**A frequency distribution shows how often each value appears in a dataset, allowing for easy visualization of data patterns. Cumulative frequency builds upon this by providing a running total of frequencies up to each value, helping illustrate how many observations fall below certain thresholds.Example in Python:**

# Frequency Distribution Example

import pandas as pd

data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]

frequency = pd.Series(data).value\_counts()

cumulative\_frequency = frequency.cumsum()

print("Frequency Distribution:\n", frequency)

print("Cumulative Frequency:\n", cumulative\_frequency)

**14. Histograms**

A histogram is a graphical representation of frequency distributions where data points are grouped into bins or intervals. It visually displays how many observations fall within each range, making it easier to identify patterns such as skewness or modality in data distributions. This structured overview provides clear explanations for each topic in basic statistics without code while ensuring each explanation contains sufficient detail for understanding fundamental concepts. If you have any further questions or need additional information on specific topics, feel free to ask!

**Example in Python:**

import matplotlib.pyplot as plt

# Histogram Example

plt.hist(data, bins=5, color='blue', edgecolor='black')

plt.title("Histogram of Data")

plt.xlabel("Data Range")

plt.ylabel("Frequency")

plt.show()

**1. Percentiles and Quantiles**

Percentiles divide data into 100 equal parts, while quantiles divide it into equal-sized intervals based on specified proportions (e.g., quartiles for 4 parts). They help in understanding the distribution of data.

python

Copy code

import numpy as np

data = [1, 2, 3, 4, 5, 6, 7, 8, 9]

percentile\_90 = np.percentile(data, 90)

quantile\_0\_25 = np.quantile(data, 0.25)

print(f"90th Percentile: {percentile\_90}, 25th Quantile: {quantile\_0\_25}")

**2. Five Number Summary**

The five-number summary includes the minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum of the dataset, giving insights into its spread.

python

Copy code

summary = {

"Min": np.min(data),

"Q1": np.percentile(data, 25),

"Median": np.median(data),

"Q3": np.percentile(data, 75),

"Max": np.max(data),

}

print(summary)

**3. Interquartile Range (IQR)**

IQR is the range between Q3 and Q1, representing the middle 50% of data and highlighting spread and outliers.

python

Copy code

iqr = np.percentile(data, 75) - np.percentile(data, 25)

print(f"Interquartile Range: {iqr}")

**4. Boxplots**

A boxplot visualizes the five-number summary and identifies outliers.

python

Copy code

import matplotlib.pyplot as plt

plt.boxplot(data)

plt.title("Boxplot")

plt.show()

**5. Effect of Outliers and Its Removal**

Outliers affect statistical measures. Removing outliers based on IQR ensures cleaner data.

python

Copy code

lower\_bound = np.percentile(data, 25) - 1.5 \* iqr

upper\_bound = np.percentile(data, 75) + 1.5 \* iqr

cleaned\_data = [x for x in data if lower\_bound <= x <= upper\_bound]

print(f"Original Data: {data}, Cleaned Data: {cleaned\_data}")

**6. Probability Density Function (PDF)**

The probability density function is essential for understanding how continuous random variables behave and helps in making informed decisions based on statistical data. It provides a framework for calculating probabilities over intervals rather than at discrete points, making it crucial for various applications in statistics and data analysis..

python

Copy code

from scipy.stats import norm

import numpy as np

x = np.linspace(-3, 3, 100)

pdf = norm.pdf(x)

plt.plot(x, pdf)

plt.title("Probability Density Function")

plt.show()

**7. Normal Distribution and Empirical Formula**

The **normal distribution**, also known as the Gaussian distribution, is a continuous probability distribution characterized by its bell-shaped curve. It is defined by two parameters: the mean (*μ*) and the standard deviation (*σ*). The normal distribution is symmetric around the mean, meaning that it has equal probabilities for values equally distant from the mean. The total area under the curve of a normal distribution equals 1, representing the total probability.

Characteristics of Normal Distribution

1. **Symmetry**: The normal distribution is symmetric about its mean.
2. **Mean, Median, and Mode**: In a normal distribution, these three measures of central tendency are equal.
3. **Empirical Rule**: Approximately 68% of data falls within one standard deviation from the mean, about 95% falls within two standard deviations, and about 99.7% falls within three standard deviations.

When a normal distribution has a mean of 0 and a standard deviation of 1, it is referred to as the **standard normal distribution**.

python

Copy code

mean, std = 0, 1

samples = np.random.normal(mean, std, 1000)

plt.hist(samples, bins=30, density=True)

plt.title("Normal Distribution")

plt.show()

**8. Z-Score**

**Definition**: A z-score (or standard score) measures how many standard deviations an individual data point is from the mean of its dataset. It indicates the relative position of a value within a distribution.**Formula**:

Z=x−μ/σ  
.

Summary of Differences

* **Variance vs. Standard Deviation**: Variance measures how far data points are from the mean but is expressed in squared units, making it less interpretable in context. Standard deviation, being the square root of variance, provides a more intuitive measure of spread in original units.
* For a population:
* σ2=∑(xi−μ)2/N
* For a sample:
* S2=∑(xi−xˉ)2/n−1
* Where:
* *σ*=sqrt(*σ*2​)
* **Standard Deviation vs. Z-Score**: Standard deviation describes overall dispersion within a dataset, while z-scores provide context for individual data points relative to that dispersion. Z-scores allow for comparison across different datasets by standardizing values based on their respective means and standard deviations.

python

Copy code

z\_scores = [(x - np.mean(data)) / np.std(data) for x in data]

print(f"Z-Scores: {z\_scores}")

**9. Standardization vs Normalization**

Standardization scales data to mean 0 and variance 1. Normalization scales data to a specific range, typically 0-1.

python

Copy code

standardized = [(x - np.mean(data)) / np.std(data) for x in data]

normalized = [(x - min(data)) / (max(data) - min(data)) for x in data]

print(f"Standardized: {standardized}, Normalized: {normalized}")

**10. Standard Normal Distribution**

The standard normal distribution has a mean of 0 and a standard deviation of 1.

python

Copy code

from scipy.stats import norm

x = np.linspace(-3, 3, 100)

pdf = norm.pdf(x)

plt.plot(x, pdf)

plt.title("Standard Normal Distribution")

plt.show()

**11. Central Limit Theorem**

The CLT states that the distribution of sample means approaches a normal distribution as the sample size increases.

The original population from which samples are drawn can have any distribution (normal, uniform, skewed, etc.). As long as the sample size is large enough, the sampling distribution of the mean will be approximately normally distributed.

python

Copy code

means = [np.mean(np.random.choice(data, 5)) for \_ in range(1000)]

plt.hist(means, bins=30)

plt.title("Central Limit Theorem")

plt.show()

**12. Chebyshev's Inequality**

* + Chebyshev's Inequality provides a way to estimate how much of our data falls within a certain distance from the average. Specifically, it tells us that no matter what kind of data we have (it could be skewed, uneven, or even normally distributed), we can make some predictions about the spread of that data.

1. **The Basic Idea**:
   * Imagine you have a group of people and you measure their heights. You find that the average height is 170 cm, but some people are much shorter or taller than this average.
   * Chebyshev's Inequality tells you that if you look at heights that are within a certain number of standard deviations (a measure of how spread out the heights are) from the average, you can expect a certain percentage of people to fall within that range.
2. **How It Works**:
   * The inequality states that for any number k*k* (which represents how many standard deviations away from the mean you're looking), at least 1−1k21−*k*21​ of the data will fall within that range.
   * For example:
     + If you look at heights within **2 standard deviations** from the average, at least **75%** of people will have heights between 150 cm and 190 cm (if we assume a standard deviation of 10 cm).
     + If you look at heights within **3 standard deviations**, at least **89%** of people will fall within that range.

python

Copy code

k = 2

chebyshev\_bound = 1 - 1 / k\*\*2

print(f"Chebyshev Bound for k=2: {chebyshev\_bound}")

**13. Covariance**

In simple terms, covariance helps us understand how two things are related: do they go up and down together (positive covariance), do they go in opposite directions (negative covariance), or do they not seem to affect each other at all (zero covariance)? It’s a useful tool for analyzing relationships in data across various fields like finance, science, and social studies.

python

Copy code

x = [1, 2, 3, 4, 5]

y = [2, 4, 6, 8, 10]

cov\_matrix = np.cov(x, y)

print(f"Covariance Matrix:\n{cov\_matrix}")

**14. Pearson Correlation**

* Pearson correlation measures the strength and direction of a linear relationship between two continuous variables. It gives a value between -1 and +1.
  + **+1** means a perfect positive linear relationship (as one variable increases, the other also increases).
  + **-1** means a perfect negative linear relationship (as one variable increases, the other decreases).
  + **0** means no linear relationship.

**Real-Time Example**:

* **Height and Weight**: Suppose you want to analyze the relationship between people's heights and weights. You collect data from several individuals:
  + As height increases, weight tends to increase as well. If you calculate the Pearson correlation coefficient and find it to be +0.85, this indicates a strong positive correlation: taller people generally weigh more.

python

Copy code

from scipy.stats import pearsonr

corr, \_ = pearsonr(x, y)

print(f"Pearson Correlation: {corr}")

**15. Spearman Correlation**

Spearman correlation measures the strength and direction of the monotonic relationship between two variables using ranked values instead of raw data. It also gives a value between -1 and +1.

* **1** indicates a perfect positive monotonic relationship.
* **-1** indicates a perfect negative monotonic relationship.
* **0** indicates no monotonic relationship.

**Real-Time Example**:

* **Class Rankings and Test Scores**: Imagine you have students who took two different tests. You want to see if there’s a relationship between their rankings in each test:
  + If students who rank high in one test also rank high in another test (even if their actual scores vary), Spearman correlation will reflect that relationship well. If you find a Spearman correlation coefficient of +0.90, it indicates a strong positive association in rankings.

python

Copy code

from scipy.stats import spearmanr

corr, \_ = spearmanr(x, y)

print(f"Spearman Correlation: {corr}")

1. **Covariance**:
   * Provides an indication of whether two variables tend to increase or decrease together.
   * The actual value can be difficult to interpret because it depends on the units of measurement.
   * Useful for understanding directional relationships but not for comparing strengths across different pairs of variables.
2. **Correlation**:
   * Offers a standardized measure that is easier to interpret and compare.
   * Indicates both the strength and direction of a linear relationship.
   * Useful for comparing relationships across various datasets or studies since it is unitless.

**1. QQ Plot: Check for Normal Distribution**

A QQ (quantile-quantile) plot compares the quantiles of a dataset against a normal distribution. If the points fall along the diagonal line, the data is normally distributed.

QQ Plot: Checking for Normal Distribution

A **QQ plot** (Quantile-Quantile plot) is a graphical tool used to assess whether a dataset follows a specific theoretical distribution, most commonly the normal distribution. Here's a detailed overview of how QQ plots work, what they indicate about normality, and how to interpret them.

What is a QQ Plot?

* A QQ plot compares the quantiles of your dataset against the quantiles of a theoretical distribution (e.g., the normal distribution).
* If the points in the QQ plot lie approximately along a straight diagonal line (from the lower left to the upper right), it suggests that the data is normally distributed.
* Deviations from this line indicate departures from normality. The nature of these deviations can provide insights into how the data differs from a normal distribution.

How to Create and Interpret a QQ Plot

1. **Creating a QQ Plot**:
   * To create a QQ plot, you typically:
     + Sort your data in ascending order.
     + Calculate the theoretical quantiles for the normal distribution based on your sample size.
     + Plot the sorted data quantiles against the theoretical quantiles.
2. **Interpreting the QQ Plot**:
   * **Points on the Line**: If most points fall on or near the diagonal line, it indicates that your data is likely normally distributed.
   * **Points Above the Line**: If points are above the line, it suggests that your data has heavier tails than a normal distribution (i.e., more extreme values).
   * **Points Below the Line**: If points fall below the line, it indicates that your data has lighter tails than a normal distribution (i.e., fewer extreme values).
   * **Curvature**: If there is noticeable curvature in the plot, it may suggest that the data is skewed or has outliers.

python

Copy code

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

data = np.random.normal(0, 1, 100)

stats.probplot(data, dist="norm", plot=plt)

plt.title("QQ Plot")

plt.show()

**2. Bernoulli Distribution and Binomial Distribution**

* **Bernoulli Distribution**: Models binary outcomes (e.g., success/failure).
* **Binomial Distribution**: Models the number of successes in nnn independent Bernoulli trials.

**Bernoulli Distribution** and **Binomial Distribution** are closely related concepts in probability theory, both dealing with experiments that have two possible outcomes. Here’s a detailed overview of each distribution and how they relate to one another.

1. Bernoulli Distribution

* **Definition**: The Bernoulli distribution is a discrete probability distribution for a random variable which takes the value 1 (success) with probability p*p* and the value 0 (failure) with probability 1−p1−*p*. It represents a single trial of a binary experiment.
* **Characteristics**:
  + **Single Trial**: The Bernoulli distribution is applicable to a single experiment or trial.
  + **Two Outcomes**: There are only two possible outcomes: success (1) and failure (0).
  + **Probability of Success**: The probability of success is denoted by p*p*, and the probability of failure is 1−p1−*p*.
* **Probability Mass Function (PMF)**:

P(X=k)=pk(1−p)1−k,k∈{0,1}*P*(*X*=*k*)=*pk*(1−*p*)1−*k*,*k*∈{0,1}

Where:

* + k=1*k*=1 for success
  + k=0*k*=0 for failure

Example of Bernoulli Distribution

* Tossing a coin once can be modeled as a Bernoulli trial where:
  + Heads (success) = 1 with probability p=0.5*p*=0.5
  + Tails (failure) = 0 with probability 1−p=0.51−*p*=0.5

2. Binomial Distribution

* **Definition**: The binomial distribution describes the number of successes in a fixed number of independent Bernoulli trials, all having the same probability of success.
* **Characteristics**:
  + **Fixed Number of Trials**: Denoted by n*n*, it represents the total number of independent trials.
  + **Binary Outcome**: Each trial results in either success or failure.
  + **Independent Trials**: The outcome of one trial does not affect the others.
  + **Constant Probability**: The probability of success p*p* remains constant across trials.
* **Probability Mass Function (PMF)**:

P(X=k)=C(n,k)pk(1−p)n−k*P*(*X*=*k*)=*C*(*n*,*k*)*pk*(1−*p*)*n*−*k*

Where:

* + C(n,k)=n!k!(n−k)!*C*(*n*,*k*)=*k*!(*n*−*k*)!*n*!​ is the number of combinations of n*n* items taken k*k* at a time.
  + k*k* is the number of successes in n*n* trials.

Example of Binomial Distribution

* Tossing a coin multiple times (e.g., flipping a coin 10 times):
  + Let’s say you want to find out how many times you get heads (success). Here, n=10*n*=10, and if the probability of getting heads in each flip is p=0.5*p*=0.5, you can use the binomial distribution to calculate the probabilities for different numbers of heads.

python

Copy code

from scipy.stats import bernoulli, binom

# Bernoulli

p = 0.5

bernoulli\_sample = bernoulli.rvs(p, size=10)

# Binomial

n, p = 10, 0.5

binomial\_sample = binom.rvs(n, p, size=10)

print(f"Bernoulli Sample: {bernoulli\_sample}")

print(f"Binomial Sample: {binomial\_sample}")

**3. Log-Normal Distribution**

A log-normal distribution is skewed and occurs when a variable’s logarithm is normally distributed.

The **log-normal distribution** is a way to describe data that can only take positive values and is often skewed to the right. Here's a simple breakdown of what it is and how it works:

python

Copy code

log\_normal\_data = np.random.lognormal(mean=0, sigma=1, size=1000)

plt.hist(log\_normal\_data, bins=30, density=True)

plt.title("Log-Normal Distribution")

plt.show()

**4. Power Law Distribution**

The power-law distribution describes phenomena with heavy tails, such as wealth distribution or natural events.

the power law distribution describes a wide range of phenomena characterized by unequal distributions where few large occurrences dominate over many smaller ones. Understanding this concept is crucial for analyzing complex systems in economics, natural sciences, and social sciences.

python

Copy code

from numpy.random import power

data = power(a=5, size=1000)

plt.hist(data, bins=30, density=True)

plt.title("Power-Law Distribution")

plt.show()

**5. Box-Cox Transform**

The Box-Cox transform makes data more normal-like by stabilizing variance and making the distribution symmetrical.

The **Box-Cox transformation** is a statistical technique used to transform non-normally distributed data into a form that is closer to a normal distribution. This transformation is particularly useful in statistical modeling and analysis, where many techniques assume that the data follows a normal distribution.

python

Copy code

from scipy.stats import boxcox

data = np.random.exponential(scale=2, size=100)

transformed\_data, \_ = boxcox(data)

plt.hist(transformed\_data, bins=30)

plt.title("Box-Cox Transformed Data")

plt.show()

**6. All Transformation Techniques**

Transformations include log, square root, reciprocal, and standardization to normalize or stabilize variance.

python

Copy code

log\_transform = np.log(data + 1)

sqrt\_transform = np.sqrt(data)

standardized = (data - np.mean(data)) / np.std(data)

print("Transformed Data Examples")

**7. Confidence Interval in Statistics**

A **confidence interval (CI)** is a statistical tool used to estimate the range within which a population parameter (like a mean or proportion) is likely to fall, based on sample data. It provides a way to express the uncertainty associated with sample estimates.python

Copy code

import scipy.stats as stats

data = np.random.normal(50, 10, 100)

mean, std = np.mean(data), np.std(data)

confidence = stats.norm.interval(0.95, loc=mean, scale=std/np.sqrt(len(data)))

print(f"95% Confidence Interval: {confidence}")

**8. Type I and Type II Error**

* **Type I Error**: Rejecting a true null hypothesis (false positive).
* **Type II Error**: Failing to reject a false null hypothesis (false negative).

python

Copy code

# Example with significance level

alpha = 0.05

print(f"Type I Error probability: {alpha}")

**9. One-Tailed and Two-Tailed Tests**

* **One-tailed**: Tests for an effect in one direction.
* **Definition**: A one-tailed test evaluates whether a parameter is either greater than or less than a certain value, focusing on one specific direction (either positive or negative).
* **Two-tailed**: Tests for an effect in both directions.
* **Definition**: A two-tailed test assesses whether a parameter is significantly different from a specified value, without specifying a direction. It checks for any change, whether an increase or decrease.

python

Copy code

from scipy.stats import ttest\_1samp

data = np.random.normal(5, 1, 30)

t\_stat, p\_value = ttest\_1samp(data, 5)

print(f"P-Value (Two-tailed): {p\_value}")

**10. Hypothesis Testing and P-Value**

Hypothesis testing uses the p-value to determine if data provides sufficient evidence to reject the null hypothesis.

Hypothesis testing is a fundamental statistical method used to make inferences about population parameters based on sample data. This process involves setting up two competing hypotheses, analyzing data, and determining whether to reject the null hypothesis based on the evidence provided by the data.

Key Concepts of Hypothesis Testing

1. **Hypotheses**:
   * **Null Hypothesis (H0)**: This is the default assumption that there is no effect or difference. It represents a statement of no change or status quo.
   * **Alternative Hypothesis (H1 or Ha)**: This hypothesis represents what you aim to prove. It suggests that there is an effect, a difference, or a relationship.
2. **Types of Tests**:
   * **One-Tailed Test**: Tests for an effect in one direction (greater than or less than).
   * **Two-Tailed Test**: Tests for any difference without specifying a direction.
3. **Test Statistic**:
   * A numerical value calculated from the sample data that is used to evaluate the hypotheses. Common test statistics include t-statistics and z-scores.
4. **P-Value**:
   * The p-value is the probability of observing the test statistic (or something more extreme) under the null hypothesis. It quantifies the evidence against H0.
   * A low p-value (typically ≤ 0.05) indicates strong evidence against the null hypothesis, leading researchers to reject H0.
   * A high p-value suggests insufficient evidence to reject H0, indicating that any observed effect could be due to random chance.
5. **Decision Making**:
   * Based on the p-value, researchers decide whether to reject or fail to reject the null hypothesis. This decision helps in making conclusions about the population based on sample data.

python

Copy code

# Hypothesis test

if p\_value < 0.05:

print("Reject the null hypothesis.")

else:

print("Fail to reject the null hypothesis.")

**11. Steps for Hypothesis Testing**

1. Define null and alternative hypotheses.
2. Choose a significance level (e.g., 0.05).
3. Compute the test statistic.
4. Calculate the p-value.
5. Make a decision (reject or fail to reject the null).

python

Copy code

# Steps demonstrated in individual tests below.

**12. T-Test**

A t-test compares the means of two groups or one group against a fixed value.

What is a T-Test?

* **Definition**: A t-test assesses whether the means of two groups are statistically different from each other. It is commonly used in hypothesis testing to evaluate the impact of a treatment or intervention.
* **Assumptions**:
  + The data should be approximately normally distributed.
  + The samples should be independent (for independent t-tests).
  + The variances of the two groups should be equal (for some types of t-tests).

Types of T-Tests

1. **One-Sample T-Test**:
   * Compares the mean of a single sample to a known value (typically the population mean).
   * **Example**: Testing if the average height of students in a class differs from the national average.
2. **Independent Two-Sample T-Test**:
   * Compares the means of two independent groups to see if they are significantly different.
   * **Example**: Comparing test scores between two different classes.
3. **Paired Sample T-Test (Dependent T-Test)**:
   * Compares means from the same group at different times (e.g., before and after a treatment).
   * **Example**: Measuring blood pressure before and after administering a medication to the same group of patients.

python

Copy code

from scipy.stats import ttest\_ind

group1 = np.random.normal(10, 2, 30)

group2 = np.random.normal(12, 2, 30)

t\_stat, p\_value = ttest\_ind(group1, group2)

print(f"T-Test p-value: {p\_value}")

**13. Z-Test**

A z-test compares sample and population means when the population standard deviation is known.

Z-tests are essential tools in statistics for hypothesis testing, particularly when dealing with larger sample sizes (typically n > 30) and when population variances are known. Understanding the different types allows researchers and analysts to choose appropriate methods for their data analysis needs.

| **Type** | **Purpose** | **Example** |
| --- | --- | --- |
| One-Sample Z-Test | Compare sample mean to population mean | Average height of students vs. national average |
| Two-Sample Z-Test | Compare means of two independent samples | Test scores between two different schools |
| Left-Tailed Z-Test | Test if parameter is less than specified value | Average score of a class is less than 75 |
| Right-Tailed Z-Test | Test if parameter is greater than specified value | Average score of a class is greater than 80 |
| Two-Tailed Z-Test | Test for any significant difference between means | Difference in average heights between two groups |
| One-Proportion Z-Test | Compare observed proportion to theoretical proportion | Proportion of voters supporting a candidate |
| Two-Proportion Z-Test | Compare proportions of two independent groups | Proportion of male vs. female students passing an exam |

python

Copy code

from statsmodels.stats.weightstats import ztest

z\_stat, p\_value = ztest(group1, group2)

print(f"Z-Test p-value: {p\_value}")

**14. ANOVA Test**

ANOVA (Analysis of Variance) compares means across three or more groups.

python

Copy code

from scipy.stats import f\_oneway

group3 = np.random.normal(15, 2, 30)

f\_stat, p\_value = f\_oneway(group1, group2, group3)

print(f"ANOVA p-value: {p\_value}")

**15. Chi-Square Test**

The chi-square test assesses the association between categorical variables.

python

Copy code

from scipy.stats import chi2\_contingency

observed = [[10, 20, 30], [20, 25, 35]]

chi2, p\_value, \_, \_ = chi2\_contingency(observed)

print(f"Chi-Square Test p-value: {p\_value}")

Let me know if you’d like to dive deeper into any of these!