Day35_Statistics_for_Machine_Learning_A_Complete_Guide

July 3, 2025

Welcome to Your Ultimate Statistics for Machine Learning Notebook

This notebook is a complete and practical guide to learning statistics with Python — designed especially for data analysts, machine learning beginners, and anyone who wants to build strong statistical intuition.

We start from the absolute basics (like mean, median, and standard deviation), and move step-by-step through probability, hypothesis testing, distributions, confidence intervals, ANOVA, regression, and more.

Each concept is explained in simple terms, supported with: - Manual calculations for better understanding - Real-world examples - Clean Python code using `NumPy`, `SciPy`, `Seaborn`, and `Pandas`

By the end of this notebook, you'll not only understand how to apply statistical methods in machine learning but also **why** they work.

Let's begin the journey from Zero to Hero in Statistics!

1 What is Statistics?

Statistics is the science of collecting, organizing, analyzing, interpreting, and presenting data. It is the foundation of data science and machine learning.

1.1 Why Statistics in Machine Learning?

Statistics is the backbone of Machine Learning. It provides the mathematical foundation to understand data, draw insights, and make predictions.

1.1.1 Understanding the Data

Before building any model, you must explore and understand the data. Statistics helps summarize and describe data through: - Mean, median, mode (central tendency) - Variance, standard deviation (dispersion) - Graphical summaries like histograms and box plots

1.1.2 Making Data-Driven Decisions

Inferential statistics allow us to: - Make predictions about a population from a sample - Use hypothesis testing to evaluate assumptions - Calculate confidence intervals to estimate possible ranges

1.1.3 Building ML Models

Many machine learning algorithms are based on statistical concepts: - Linear Regression: uses least squares estimation - Logistic Regression: based on probability and odds - Naive Bayes: uses Bayes' Theorem - Clustering and PCA: use variance, covariance, and distributions

1.1.4 Evaluating Model Performance

Statistics helps us: - Analyze model accuracy - Calculate error metrics like MAE, RMSE, R^2 - Perform A/B testing for model comparison

1.1.5 Avoiding Bias and Overfitting

Statistical tests help: - Validate model assumptions - Detect data skew or imbalance - Choose the right features and techniques

1.1.6 Summary

Without statistics, machine learning becomes a black box. With statistics, you understand why your model behaves the way it does and how to improve it.

2 Types of Statistics

2.1 Descriptive Statistics

Definition: Used to summarize and describe the features of a dataset. **Goal:** Understand the basic features of data. **Includes:** Mean, Median, Mode, Range, Variance, Standard Deviation.

2.1.1 Measures of Central Tendency

- Mean
- Median
- Mode ### Measures of Dispersion
- Range
- Variance
- Standard Deviation

Manual Example:

Given Data = [10, 12, 11, 13, 10]

Step 1: Sort the data: [10, 10, 11, 12, 13]

Mean = (10 + 12 + 11 + 13 + 10) / 5 = 56 / 5 = 11.2

Median = Middle value = 11 (3rd value in sorted list)

Mode = Value that appears most often = 10 (appears twice)

Range = Max - Min = 13 - 10 = 3

Variance:

- Find the mean = 11.2
- Subtract mean from each value: [-1.2, 0.8, -0.2, 1.8, -1.2]
- Square differences: [1.44, 0.64, 0.04, 3.24, 1.44]
- Average of squared differences: (1.44 + 0.64 + 0.04 + 3.24 + 1.44) / 5 = 1.36

Standard Deviation = Square root of Variance = sqrt(1.36) 1.166

[42]:
```
import numpy as np
import statistics as stats

data = [10, 12, 11, 13, 10]
print("Mean:", np.mean(data))
print("Median:", np.median(data))
print("Mode:", stats.mode(data))
print("Range:", max(data) - min(data))
print("Variance:", np.var(data))
print("Standard Deviation:", np.std(data))
```

```
Mean: 11.2
Median: 11.0
Mode: 10
Range: 3
Variance: 1.3599999999999999
Standard Deviation: 1.16619037896906
```

3 Inferential Statistics

Definition: Makes predictions or inferences about a population using a sample.

Goal: Generalize findings from sample to population.

Includes: Hypothesis testing, Confidence Intervals, Regression, ANOVA, Chi-square.

4 Population and Sampling

Understanding the difference between a population and a sample is essential in statistics, especially for drawing valid conclusions in machine learning projects.

4.1 Population vs Sample

- **Population**: The entire group you're interested in studying.
  - Example: All users on Amazon in India.
- **Sample**: A subset of the population that is actually observed or analyzed.
  - Example: 1,000 Amazon users selected randomly from Pune.

We use **samples** because it is usually not possible (or practical) to collect data from an entire population. Statistics helps us use the sample to make inferences about the whole.

4.2 Types of Sampling Techniques

There are several ways to choose a sample. Here are the most common types:

4.2.1 Simple Random Sampling

- Every member of the population has an equal chance of being selected.

Manual Example: If there are 100 students, and we randomly pick 10 using a lottery draw.

[43]:
```
import random

population = list(range(1, 101))  # 100 students numbered 1 to 100
sample = random.sample(population, 10)
print("Random Sample:", sample)
```

```
Random Sample: [12, 57, 16, 80, 91, 19, 83, 22, 68, 33]
```

4.2.2 Stratified Sampling

Divide the population into subgroups (strata) and randomly sample from each group.

Manual Example: Separate male and female students, then take 5 random from each.

[44]:
```
import random

males = ['M1', 'M2', 'M3', 'M4', 'M5', 'M6']
females = ['F1', 'F2', 'F3', 'F4', 'F5', 'F6']

sample_males = random.sample(males, 3)
sample_females = random.sample(females, 3)

print("Stratified Sample:", sample_males + sample_females)
```

```
Stratified Sample: ['M5', 'M4', 'M3', 'F5', 'F6', 'F4']
```

4.2.3 Cluster Sampling

Divide the population into clusters, then randomly choose entire clusters.

Manual Example: Choose 2 out of 10 schools and survey all students in those schools.

[45]:
```
import random

clusters = {
    'School A': ['A1', 'A2', 'A3'],
    'School B': ['B1', 'B2', 'B3'],
    'School C': ['C1', 'C2', 'C3']
}

selected_clusters = random.sample(list(clusters.keys()), 2)
sample = []
```

```
for cluster in selected_clusters:
    sample.extend(clusters[cluster])

print("Cluster Sample:", sample)
```

```
Cluster Sample: ['A1', 'A2', 'A3', 'C1', 'C2', 'C3']
```

4.2.4 Systematic Sampling

Select every k-th member from the list.

Manual Example: From a list of 100, choose every 10th person.

[46]:
```
population = list(range(1, 101))
k = 10
sample = population[::k]  # Select every 10th element
print("Systematic Sample:", sample)
```

```
Systematic Sample: [1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
```

4.2.5 Convenience Sampling

Select individuals that are easiest to reach.

Manual Example: Ask people in your neighborhood or classroom.

[47]:
```
# Convenience sampling - just select first few available
population = list(range(1, 21))
convenient_sample = population[:5]  # pick first 5 available
print("Convenience Sample:", convenient_sample)
```

```
Convenience Sample: [1, 2, 3, 4, 5]
```

Summary

Sampling techniques determine how representative your data is. Poor sampling can lead to biased models. Always try to use random or stratified sampling when possible in real-world ML problems.

5 Probability Basics

Probability tells us how likely an event is to happen. It is one of the core building blocks of statistics and machine learning.

5.1 Basic Formula

Probability of an event (A) is calculated as:

Probability (A) = Number of favorable outcomes / Total number of outcomes

- The result is always between 0 and 1.
  - 0 means impossible

– 1 means certain

Example: If we roll a fair 6-sided die, the probability of getting a 4 is:

[19]:
```
favorable_outcomes = 1
total_outcomes = 6
probability = favorable_outcomes / total_outcomes
print("Probability of rolling a 4:", probability)
```

```
Probability of rolling a 4: 0.16666666666666666
```

5.2 Real-World Examples

Example 1: Coin Toss - Event: Getting heads

- Possible outcomes: Heads, Tails

P(Heads) = 1 / 2 = 0.5

[21]:
```
print("Probability of Heads:", 1/2)
```

```
Probability of Heads: 0.5
```

Example 2: Drawing a Red Card from a Deck - Total cards = 52

- Red cards (hearts + diamonds) = 26

P(Red card) = 26 / 52 = 0.5

[22]:
```
red_cards = 26
total_cards = 52
print("Probability of red card:", red_cards / total_cards)
```

```
Probability of red card: 0.5
```

Example 3: Picking a Vowel from a List of Letters - Letters = [‘A’, ‘B’, ‘C’, ‘E’, ‘I’, ‘J’]

- Vowels = A, E, I → 3 out of 6

P(Vowel) = 3 / 6 = 0.5

[23]:
```
letters = ['A', 'B', 'C', 'E', 'I', 'J']
vowels = ['A', 'E', 'I']
probability = len(vowels) / len(letters)
print("Probability of picking a vowel:", probability)
```

```
Probability of picking a vowel: 0.5
```

Summary

Probability helps us reason under uncertainty. It’s used in:

Risk prediction

Randomized experiments

Naive Bayes classifiers

Evaluation metrics (like expected accuracy)

Understanding it is key for both theory and practical machine learning.

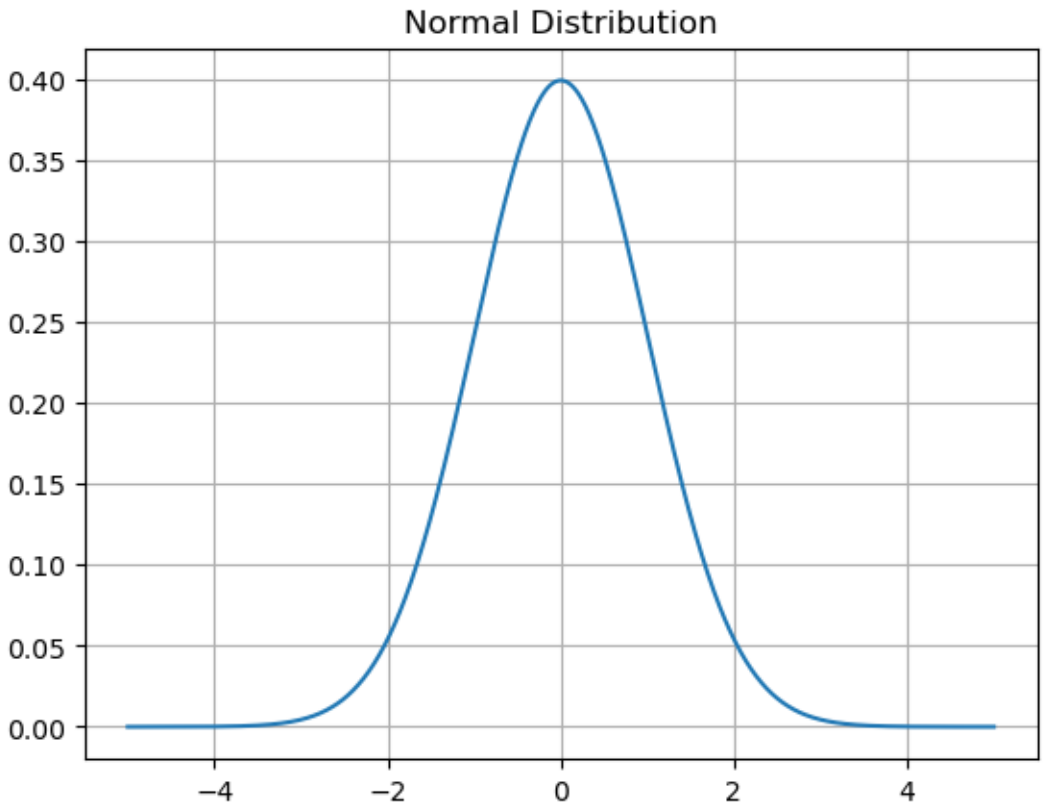
6 Distributions

Understanding distributions is key to identifying patterns in data and choosing the right analysis method.

6.1 Normal Distribution

- **Also called**: Gaussian Distribution
- **Shape**: Bell-shaped, symmetric
- **Mean = Median = Mode**
- **Use when**: Data is naturally centered around a mean (e.g., heights, test scores)
- **Real-life examples**:
  - Heights of people
  - Exam scores in large groups
  - Errors in measurement
- **In ML**: Used in linear regression assumptions, hypothesis testing, z-tests

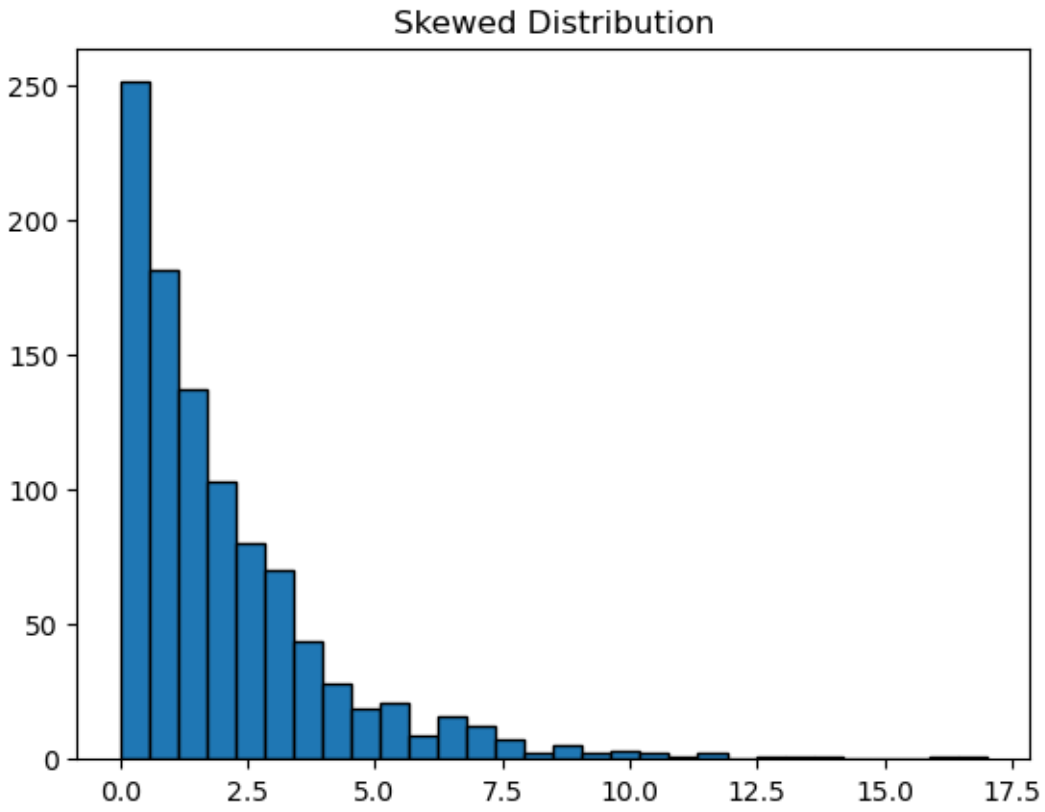
[24]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 1000)
y = 1/(np.sqrt(2*np.pi)) * np.exp(-x**2/2)

plt.plot(x, y)
plt.title("Normal Distribution")
plt.grid(True)
plt.show()
```



7 Skewed Distribution

- **Shape**: Asymmetric
  - **Positively skewed**: Long tail to the right (e.g., income data)
  - **Negatively skewed**: Long tail to the left
- **Mean Median Mode**
- **Use when**: Data has outliers or is not balanced
- **Real-life examples**:
  - House prices (a few very expensive houses skew the average)
  - Income distribution
- **In ML**: You may need to apply transformations (like log) before modeling

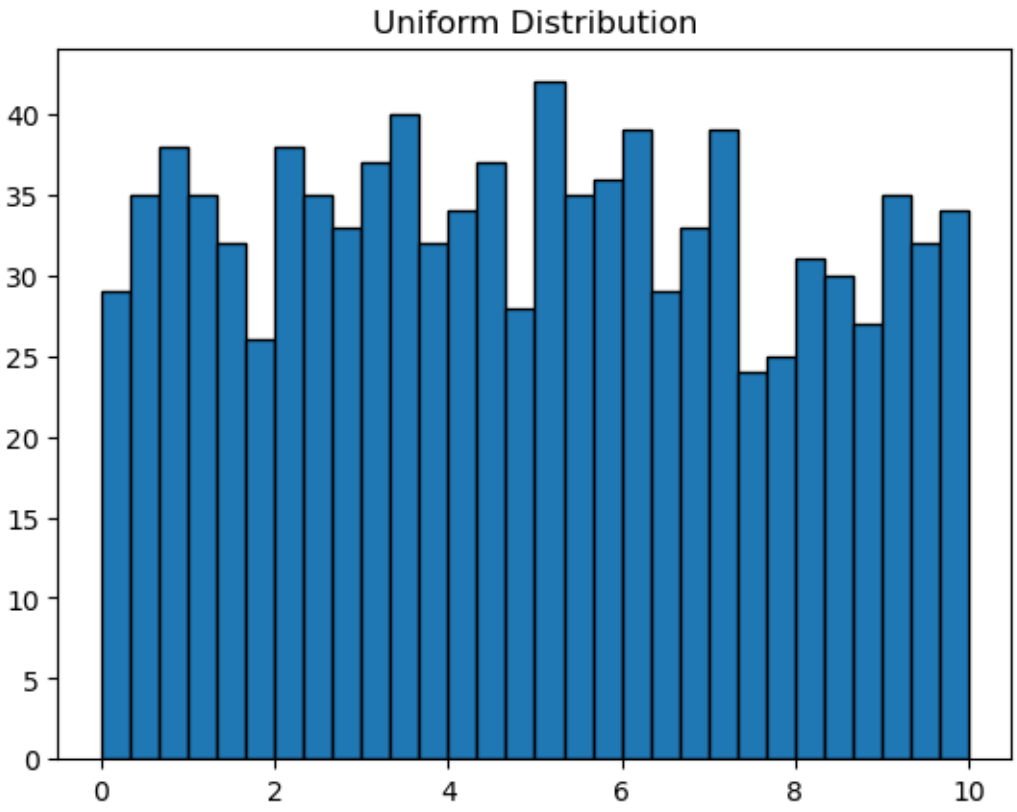
[26]:
```
skewed_data = np.random.exponential(scale=2, size=1000)
plt.hist(skewed_data, bins=30, edgecolor='black')
plt.title("Skewed Distribution")
plt.show()
```



7.1 Uniform Distribution

- **Shape**: All outcomes are equally likely (flat shape)
- **Use when**: Every item has an equal chance of being selected
- **Real-life examples**:
  - Rolling a fair die
  - Random number generation
- **In ML**: Used in simulations, initialization of weights in neural networks

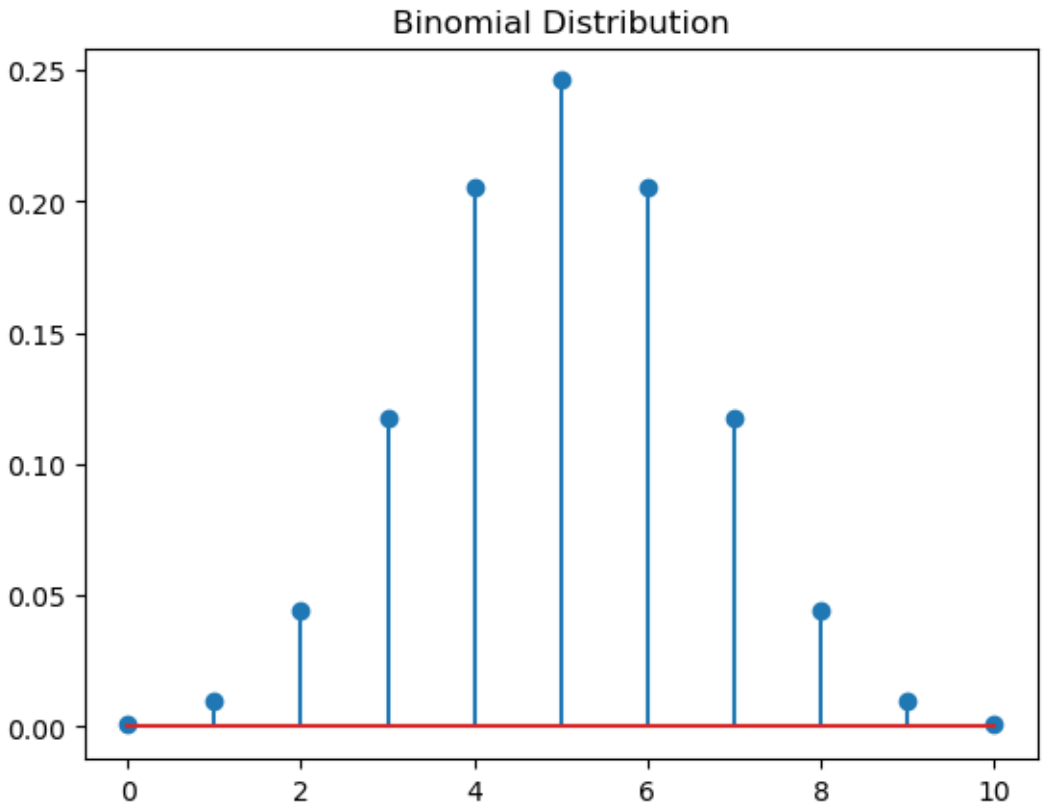
[27]:
```
uniform_data = np.random.uniform(0, 10, 1000)
plt.hist(uniform_data, bins=30, edgecolor='black')
plt.title("Uniform Distribution")
plt.show()
```



7.2 Binomial Distribution

- **Type**: Discrete distribution
- **Used for**: Counting the number of successes in a fixed number of trials
- **Conditions**:
  - Only 2 possible outcomes (Success/Failure)
  - Fixed number of trials
  - Each trial is independent
- **Real-life examples**:
  - Tossing a coin 10 times
  - Predicting the number of defective products in a batch
- **In ML**: Used in binary classification, logistic regression, A/B testing

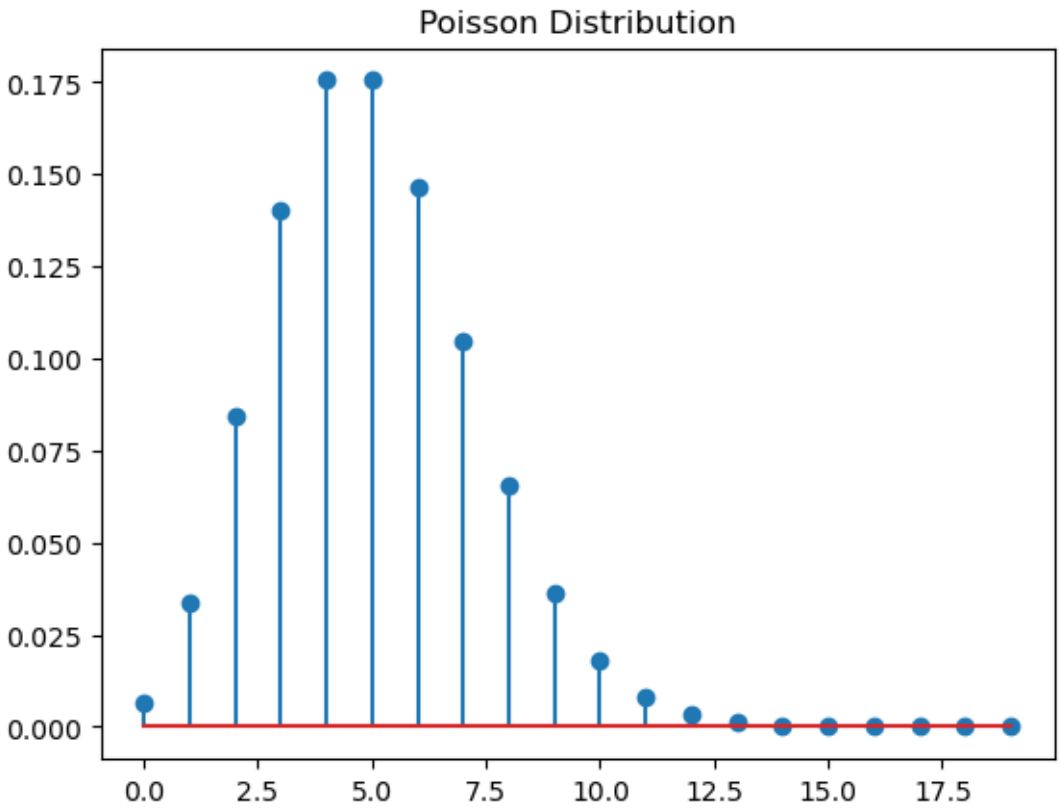
[28]:
```
from scipy.stats import binom
x = np.arange(0, 11)
y = binom.pmf(x, 10, 0.5)
plt.stem(x, y)
plt.title("Binomial Distribution")
plt.show()
```



7.3 Poisson Distribution

- **Type**: Discrete distribution
- **Used for**: Counting number of events happening in a fixed interval of time or space
- **Conditions**:
  - Events happen independently
  - Events are rare
- **Real-life examples**:
  - Number of customer calls per hour
  - Number of earthquakes in a year
- **In ML**: Used in modeling rare events, queuing systems, count data

[29]:
```
from scipy.stats import poisson
x = np.arange(0, 20)
y = poisson.pmf(x, 5)
plt.stem(x, y)
plt.title("Poisson Distribution")
plt.show()
```



8 Hypothesis Testing

Hypothesis testing is used to determine whether the results of a dataset are statistically significant and not due to random chance.

8.1 Concepts

- **Null Hypothesis (H0)**: There is no effect or difference. It represents the default assumption.
  - Example: The average score of students is 70.
- **Alternative Hypothesis (H1 or Ha)**: There is an effect or a difference.
  - Example: The average score is not 70.

We perform a test to decide whether to reject H0 or not.

8.2 Steps of Hypothesis Testing

1. Define H0 and H1.
2. Choose significance level (usually 0.05).
3. Select a test (Z-test, T-test, etc.).
4. Calculate the test statistic.
5. Calculate or look up the p-value.

6. Compare p-value with significance level:
    - If p-value $< 0.05 \rightarrow$ Reject H0.
    - If p-value 0.05 $\rightarrow$ Fail to reject H0.

8.3 Manual Example (T-test)

Suppose the average test score (population mean) is 70. We test a new teaching method on 10 students. Their scores are:

[72, 75, 68, 70, 69, 74, 73, 71, 67, 76]

Is the new method significantly different?

- H0: Mean = 70
- H1: Mean 70
- Significance level = 0.05

Steps:

1. Calculate sample mean and sample standard deviation.
2. Use the t-test formula:

Example (T-test):

[30]:
```
from scipy import stats

scores = [72, 75, 68, 70, 69, 74, 73, 71, 67, 76]
t_stat, p_val = stats.ttest_1samp(scores, 70)

print("T-statistic:", t_stat)
print("P-value:", p_val)

if p_val < 0.05:
    print("Reject Null Hypothesis")
else:
    print("Fail to Reject Null Hypothesis")
```

```
T-statistic: 1.5666989036012806
P-value: 0.1516274744876827
Fail to Reject Null Hypothesis
```

9 Z-Test vs T-Test

Z-test and T-test are both hypothesis tests used to determine whether a sample mean is significantly different from a known or assumed population mean.

9.1 What is a Z-Test?

Z-test is used when: - Sample size is **large (n $>$ 30)** - **Population standard deviation is known**

It assumes a **normal distribution**.

Formula:

Z = ($\bar{x}$ -) / (/ $\sqrt{n}$)

Where: - $\bar{x}$ = sample mean
- = population mean
- = population standard deviation
- n = sample size

Manual Example (Z-Test):

A factory claims that the average weight of their packets is 500g. A sample of 50 packets has a mean of 495g. Population standard deviation is known to be 20g.

- H0: = 500
- H1: 500
- Significance level = 0.05

Steps: - Sample mean = 495
- Population mean = 500
- = 20
- n = 50

Z = (495 - 500) / (20 / $\sqrt{50}$) -1.77

Critical z-values for 95% CI: ±1.96 → Since -1.77 is within range, **Fail to Reject H0**

[31]:
```
# Python Example (Z-Test):

import statsmodels.stats.weightstats as stests

data = [495] * 50  # simulate sample
ztest, pval = stests.ztest(data, value=500)
print("Z-statistic:", ztest)
print("P-value:", pval)
```

```
Z-statistic: -inf
P-value: 0.0
```

```
C:\Users\Lenovo\anaconda3\Lib\site-
packages\statsmodels\stats\weightstats.py:748: RuntimeWarning: divide by zero
encountered in scalar divide
  zstat = (value1 - value2 - diff) / std_diff
```

9.2 T-Test (One-Sample)

A T-test is a statistical test used to determine whether the mean of a sample differs significantly from a known or assumed population mean.

9.3 When to Use a T-Test?

Use a **T-test** when: - The **sample size is small (n < 30)** - The **population standard deviation is unknown** - Data is approximately normally distributed

It uses the **Student's t-distribution**, which has heavier tails than the normal distribution to account for more variability in small samples.

Formula

t = ($\bar{x}$ -) / (s / $\sqrt{n}$)

Where: - $\bar{x}$ = sample mean
- = population mean
- s = sample standard deviation
- n = sample size

Manual Example

Suppose a teacher claims the average score is 70. We want to test this claim using the following 10 student scores:

[72, 75, 68, 70, 69, 74, 73, 71, 67, 76]

- H0: = 70 (No difference)
- H1: 70 (There is a difference)
- Significance level = 0.05

Steps: 1. Calculate sample mean and standard deviation. 2. Plug values into the t formula. 3. Compare the result with the critical value from the t-distribution table. 4. Or simply calculate the p-value.

[33]:
```
from scipy import stats

scores = [72, 75, 68, 70, 69, 74, 73, 71, 67, 76]
t_stat, p_val = stats.ttest_1samp(scores, 70)

print("T-statistic:", t_stat)
print("P-value:", p_val)

if p_val < 0.05:
    print("Reject Null Hypothesis")
else:
    print("Fail to Reject Null Hypothesis")
```

```
T-statistic: 1.5666989036012806
P-value: 0.1516274744876827
Fail to Reject Null Hypothesis
```

9.4 Interpretation

- If p-value < 0.05 → Reject H0 → There is a statistically significant difference.
- If p-value 0.05 → Fail to reject H0 → No significant difference.

9.5 When to Use Other Types of T-Tests?

- One-sample T-test: Compare sample mean to known value (as shown above)
- Two-sample T-test: Compare means of two independent groups
- Paired T-test: Compare means from the same group at different times

Summary - T-tests are powerful tools for small-sample inference.

- Always check whether your data meets assumptions (normality, independence).
- Use one-sample t-tests when comparing a group mean to a known or target value.

10 Confidence Intervals

A confidence interval gives a range of values that is likely to contain a population parameter (such as the mean) with a certain level of confidence (e.g., 95%).

10.1 Definition and Purpose

- A **Confidence Interval (CI)** estimates the range in which the true population mean likely falls.
- It is expressed as:

$CI = \bar{x} \pm z * (s / \sqrt{n})$

Where: - $\bar{x}$ = sample mean
- z = z-score for the desired confidence level (e.g., 1.96 for 95%)
- s = sample standard deviation
- n = sample size

- The **confidence level** (commonly 90%, 95%, or 99%) tells how sure we are the true mean lies within the range.

10.2 Manual Calculation Example

Let's say we have the following:

- Sample mean ($\bar{x}$) = 100
- Sample standard deviation (s) = 15
- Sample size (n) = 25
- Confidence level = 95% $\rightarrow$ z = 1.96

Step-by-step:

1. Calculate standard error (SE):
   $SE = s / \sqrt{n} = 15 / \sqrt{25} = 15 / 5 = 3$
2. Calculate margin of error (ME):
   $ME = z * SE = 1.96 * 3 = 5.88$

3. Calculate confidence interval:
   $\text{CI} = 100 \pm 5.88 \rightarrow [94.12, 105.88]$

Interpretation: We are 95% confident that the true population mean lies between 94.12 and 105.88.

[35]:
```python
import numpy as np
import scipy.stats as stats

sample = [100, 102, 98, 101, 97, 99, 100, 102, 98, 103]

mean = np.mean(sample)
std_err = stats.sem(sample)  # Standard Error
confidence_level = 0.95

# t-distribution CI since sample std dev is used
ci = stats.t.interval(confidence_level, len(sample)-1, loc=mean, scale=std_err)

print("Sample Mean:", mean)
print("95% Confidence Interval:", ci)
```

```
Sample Mean: 100.0
95% Confidence Interval: (np.float64(98.56928618802333),
np.float64(101.43071381197667))
```

11 ANOVA (Analysis of Variance)

ANOVA (Analysis of Variance) is a statistical method used to compare the means of **three or more independent groups** to determine if at least one group mean is significantly different from the others.

11.1 Why Use ANOVA?

- When you have **more than two groups**, doing multiple t-tests increases the risk of Type I error (false positive).
- ANOVA solves this by testing **all group means simultaneously** with **just one test**.

11.2 Example Scenario

Suppose you are testing the effect of three different diets on weight loss.

- Group A: High protein
- Group B: Low fat
- Group C: Balanced diet

You want to know: **Do these diets lead to different average weight losses?**

Instead of comparing A vs B, A vs C, and B vs C with multiple t-tests, you run **one ANOVA test** to check if there's a significant difference **among all three groups**.

11.3 What ANOVA Tells You

- **Null Hypothesis (H0)**: All group means are equal
- **Alternative Hypothesis (H1)**: At least one group mean is different

If the p-value < 0.05, we reject H0 $\rightarrow$ This means **at least one group is significantly different**.

11.4 Important Notes

- ANOVA tells you **if there's a difference**, but **not which groups are different**.
- If ANOVA is significant, we follow it with **post-hoc tests** (like Tukey's test) to find where the differences lie.

[5]:
```
import scipy.stats as stats

group1 = [23, 45, 67, 89]
group2 = [24, 47, 69, 90]
group3 = [22, 43, 66, 85]

f_stat, p_val = stats.f_oneway(group1, group2, group3)
print("F-statistic:", f_stat)
print("p-value:", p_val)
```

```
F-statistic: 0.015653645466083766
p-value: 0.9844949778346346
```

11.5 Summary

Use ANOVA when: - You want to compare **three or more group means** - Your data is continuous and normally distributed - The groups are independent and have roughly equal variances

ANOVA is commonly used in experiments, A/B testing, clinical trials, and machine learning model comparison.

12 Chi-Square Test

The **Chi-Square Test** is a statistical test used to determine whether there is a **significant association between two categorical variables**.

12.1 Why Use a Chi-Square Test?

Use it when: - Your data is **categorical** (e.g., gender, preferences, yes/no, types) - You want to know if two variables are **independent** or **related**

It compares the **observed frequencies** (what actually happened) with the **expected frequencies** (what you would expect if there were no association).

12.2 Types of Chi-Square Tests

1. **Chi-Square Test of Independence**
   - Used to check if two categorical variables are related.
   - Example: Is there a relationship between gender and product preference?
2. **Chi-Square Goodness-of-Fit Test**
   - Used to check if an observed distribution fits a theoretical one.
   - Example: Do coin toss outcomes match the expected 50-50 split?

12.3 Example Scenario

You survey 100 people about their favorite ice cream flavor (Vanilla, Chocolate, Strawberry) and also note their gender.

You want to know:
Is ice cream preference related to gender?

You organize the responses in a contingency table, then run a **Chi-Square Test of Independence**.

12.4 What Chi-Square Tells You

- **Null Hypothesis (H0)**: The two variables are independent (no association)
- **Alternative Hypothesis (H1)**: The two variables are dependent (there is an association)

If the p-value $< 0.05 \rightarrow$ You reject H0 $\rightarrow$ There is a significant relationship between the two variables.

[6]:
```
import pandas as pd
import scipy.stats as stats

data = [[30, 10], [20, 40]]
chi2, p, dof, expected = stats.chi2_contingency(data)
print("Chi2 Statistic:", chi2)
print("p-value:", p)
```

```
Chi2 Statistic: 15.041666666666666
p-value: 0.00010516355403363098
```

12.5 Summary

Use the Chi-Square Test when: - You are working with **categorical data** - You want to check **relationships between variables** - You have data organized in a **contingency table**

Chi-Square is often used in: - Market research (e.g., preference vs region) - Medical studies (e.g., treatment vs recovery) - A/B testing (e.g., version A vs B click-through rates)

13 Correlation

Correlation is a statistical measure that describes the **strength** and **direction** of a relationship between two numerical (continuous) variables.

13.1 Key Points

- Correlation values range from **-1 to +1**.

Correlation Value	Meaning
+1	Perfect positive correlation
0	No correlation
-1	Perfect negative correlation

- A **positive correlation** means that as one variable increases, the other tends to increase as well.
- A **negative correlation** means that as one variable increases, the other tends to decrease.
- A **zero correlation** means there is no relationship between the two variables.

13.2 Example Interpretations

- Height and weight: **Positive correlation** (taller people tend to weigh more)
- Exercise time and body fat: **Negative correlation** (more exercise, less fat)
- Roll of a dice and day of the week: **No correlation**

13.3 Important Notes

- Correlation **does not imply causation**.
  - Just because two variables move together doesn't mean one causes the other.
- Correlation is useful for:
  - **Feature selection** in machine learning
  - **Detecting multicollinearity**
  - **Exploratory Data Analysis (EDA)**

[7]:
```
import numpy as np
x = [1, 2, 3, 4, 5]
y = [2, 4, 5, 4, 5]
correlation = np.corrcoef(x, y)
print("Correlation Matrix:\n", correlation)
```

```
Correlation Matrix:
 [[1.         0.77459667]
 [0.77459667 1.        ]]
```

13.4 Summary

Use correlation to: - Measure how strongly two variables are related - Understand relationships before modeling - Choose or remove features in machine learning

14 Linear Regression

Linear Regression is one of the most fundamental and widely used algorithms in statistics and machine learning. It helps us understand the **relationship between two variables** by fitting a straight line through the data.

14.1 Introduction to Regression

Regression is a predictive modeling technique that estimates the relationship between a **dependent variable (target)** and one or more **independent variables (features)**.

In **Simple Linear Regression**, we predict the dependent variable (Y) using just one independent variable (X).

The model assumes a linear relationship: Y = mX + c

Where: - Y = predicted value (dependent variable) - X = input variable (independent variable) - m = slope of the line - c = y-intercept Use Cases of Linear Regression

- Predicting sales based on advertising spend
- Estimating house prices based on size
- Predicting temperature based on time of day

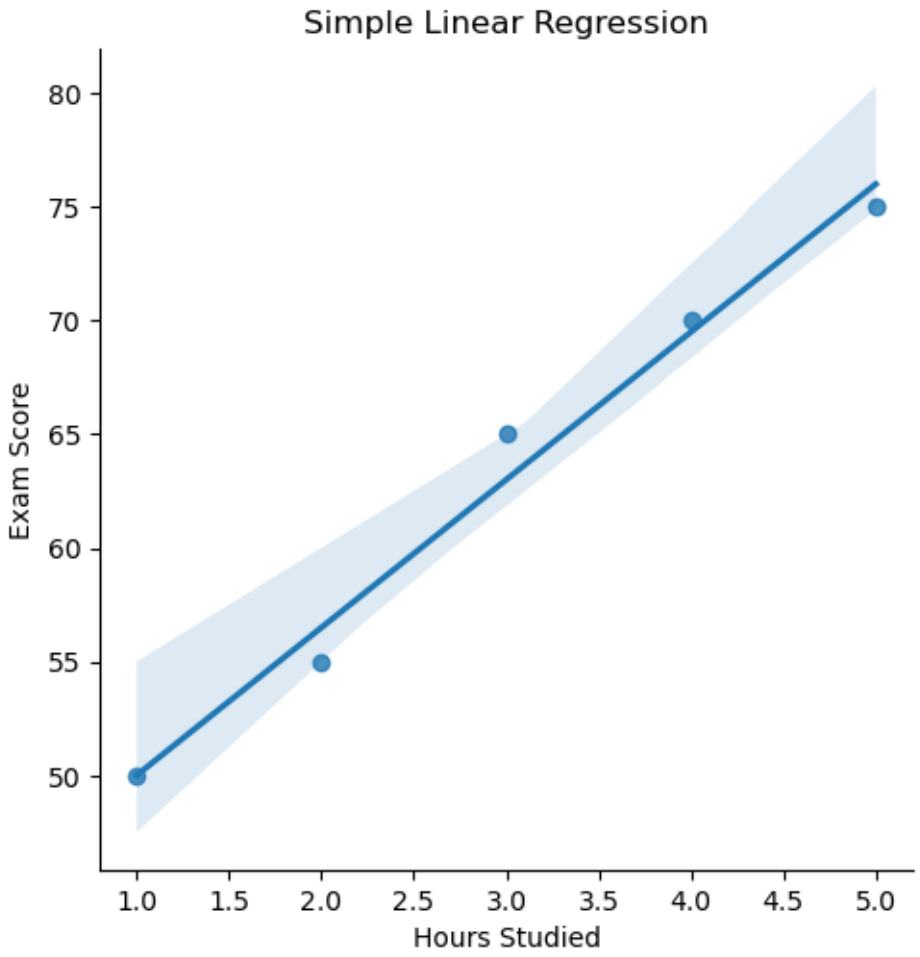
Simple Linear Regression Example

Imagine we have data showing how the number of study hours affects exam scores:

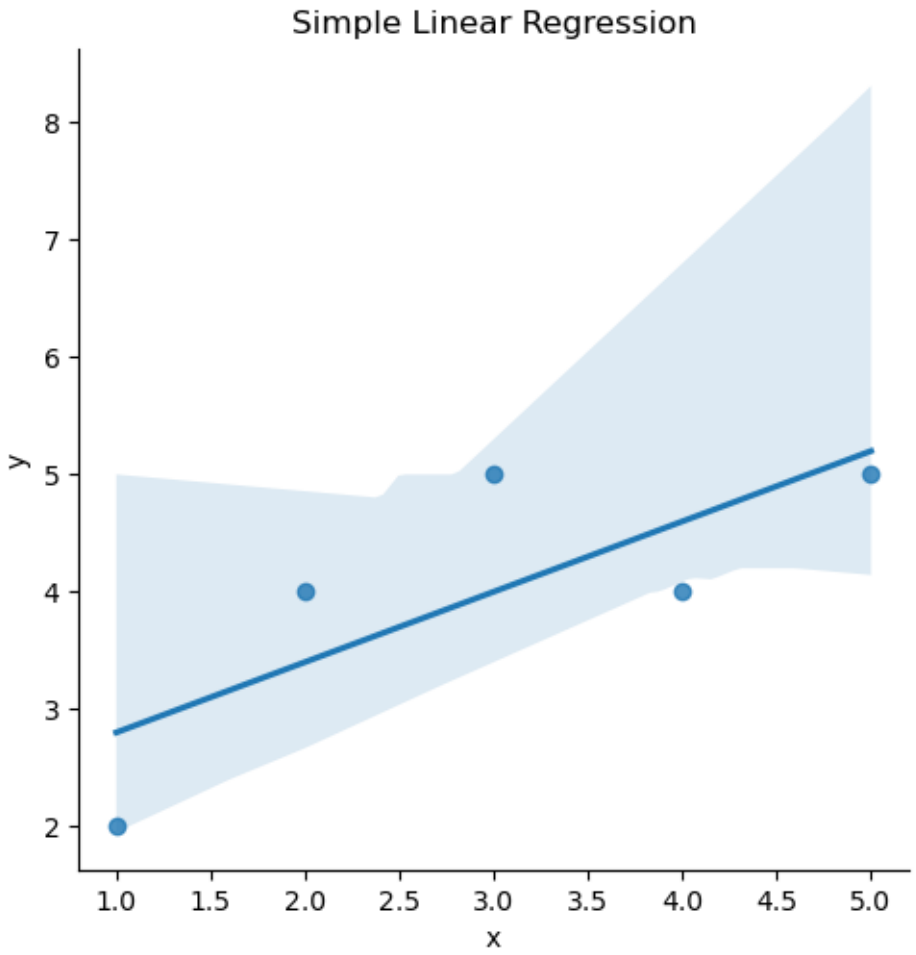
Hours	Score
1	50
2	55
3	65
4	70
5	75

We can use linear regression to **fit a line** that predicts score based on hours studied.

[37]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = {
    'Hours': [1, 2, 3, 4, 5],
    'Score': [50, 55, 65, 70, 75]
}
df = pd.DataFrame(data)

# Create a linear regression plot
sns.lmplot(x='Hours', y='Score', data=df)
plt.title("Simple Linear Regression")
plt.xlabel("Hours Studied")
plt.ylabel("Exam Score")
plt.show()
```


Simple Linear Regression
Exam Score
80
75
70
65
60
55
50
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Hours Studied

[8]:



14.2 Summary

- Linear regression models the relationship between two variables using a straight line.
- It's easy to interpret and fast to compute.
- Commonly used for prediction, trend analysis, and feature importance.

15 Multiple Linear Regression and Model Evaluation

15.1 What is Multiple Linear Regression?

Multiple Linear Regression is an extension of simple linear regression. Instead of using one independent variable (X), it uses **two or more** variables to predict the dependent variable (Y).

Equation: Y = b0 + b1X1 + b2X2 + … + bn*Xn

Where: - Y = target variable (dependent) - X1, X2, …, Xn = input variables (independent) - b0 = intercept - b1, b2, …, bn = coefficients

15.2 When to Use It?

Use multiple linear regression when: - The outcome depends on more than one factor - You want to evaluate the impact of multiple variables at once

15.3 Example Scenario

You want to predict the **house price** based on: - Square footage - Number of bedrooms - Age of the house

[38]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Example dataset
data = {
    'Area': [1000, 1500, 1700, 1300, 1600],
    'Bedrooms': [2, 3, 3, 2, 3],
    'Age': [10, 5, 3, 8, 4],
    'Price': [300000, 400000, 420000, 330000, 410000]
}
df = pd.DataFrame(data)

# Features and target
X = df[['Area', 'Bedrooms', 'Age']]
y = df['Price']

# Train model
model = LinearRegression()
model.fit(X, y)

# Predict
predictions = model.predict(X)
# Show coefficients
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
```

```
Intercept: 100000.00000000029
Coefficients: [ 1.00000000e+02  5.00000000e+04 -1.29153671e-11]
```

[ ]:
```
## Evaluation Metrics for Regression
After fitting a regression model, we use evaluation metrics to assess how well␣
 ↪the model is performing.

### R² Score (Coefficient of Determination)
- Indicates how much of the variance in Y is explained by X

- Value ranges from 0 to 1 (higher is better)
```

[39]:
```
r2 = r2_score(y, predictions)
print("R² Score:", r2)
```

```
R² Score: 1.0
```

15.3.1 Mean Absolute Error (MAE)

- Average of absolute errors between predicted and actual values
- Lower MAE means better accuracy

[40]:
```
mae = mean_absolute_error(y, predictions)
print("Mean Absolute Error:", mae)
```

```
Mean Absolute Error: 0.0
```

15.3.2 Mean Squared Error (MSE)

- Average of squared errors
- More sensitive to outliers than MAE

[41]:
```
mse = mean_squared_error(y, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.0
```

15.4 Summary

- Multiple Linear Regression is used when there are multiple input features.
- Always evaluate the model using metrics like R^2, MAE, and MSE.
- Higher R^2 and lower MAE/MSE indicate a better-performing regression model.