Department of Artificial Intelligence and Data Science

Fundamentals of Data Science (AD45)

Academic Year : 2024-24, Batch 2023

Credits: 3:0:0

Text Books:

Joel Grus, "Data Science from Scratch", 2nd Edition, O'Reilly Publications/Shroff Publishers and Distributors Pvt. Ltd., 2019. ISBN-13: 978- 9352138326

UNIT 1

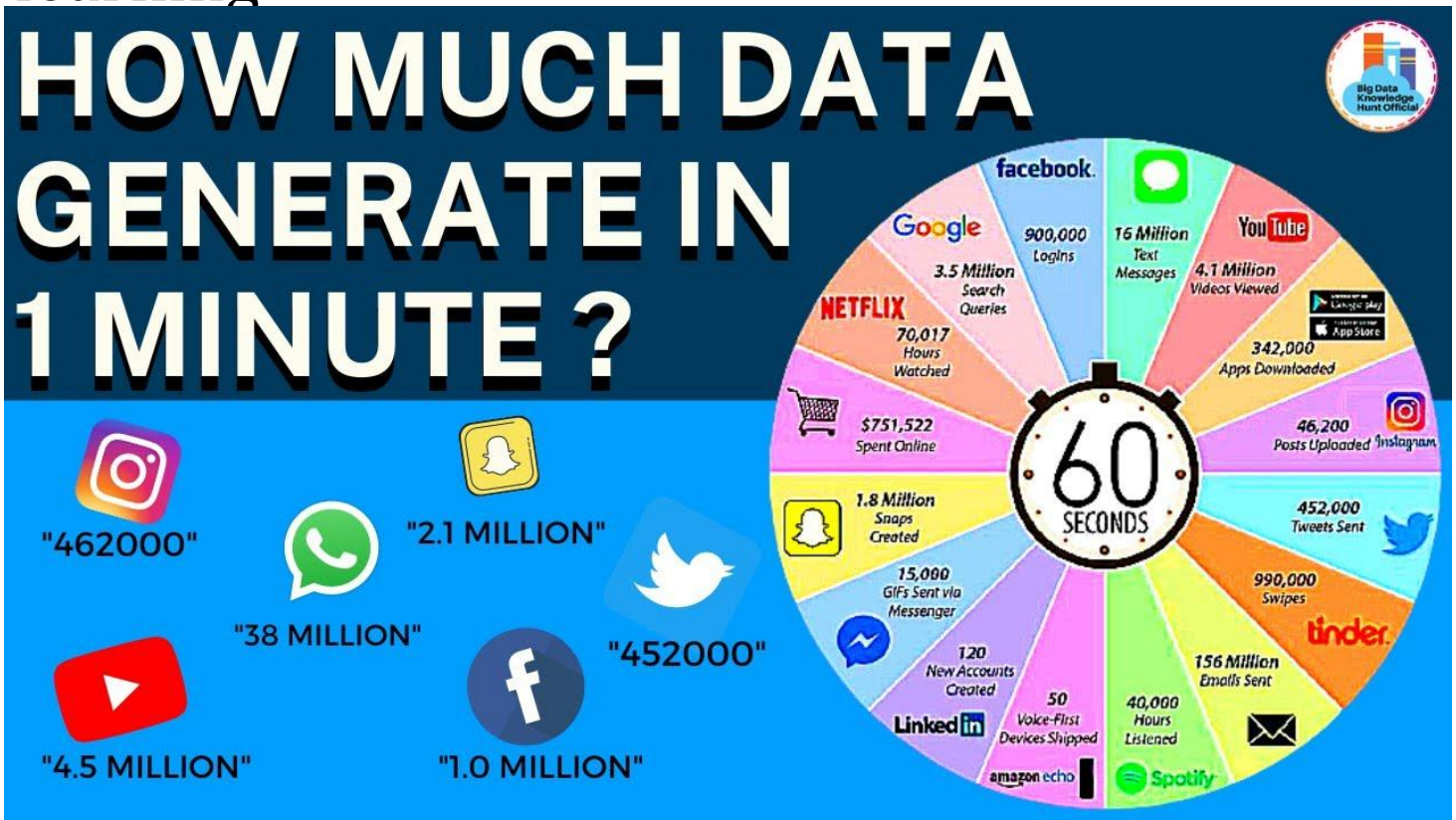
What is Data Science? Types of Data & Data Sources, Visualizing Data, matplotlib, Bar Charts, Line Charts, Scatterplots, Linear Algebra, Vectors, Matrices, Statistics, Describing a Single Set of Data, Correlation, Simpson's Paradox, Some Other Correlational Caveats, Correlation and Causation. Probability, Dependence and Independence, Random Variables, Continuous Distributions, The Normal Distribution.

The Ascendance of Data

- We live in a world that's drowning in data. Websites track every user's every click.
- Your **smartphone** is building up a record of your location and speed every second of every day. "Quantified selfers" wear pedometers-on-steroids that are always recording their heart rates, movement habits, diet, and sleep patterns.
- **Smart cars** collect driving habits, smart homes collect living habits, and smart marketers collect purchasing habits.
- The **internet** itself represents a huge graph of knowledge that contains (among other things) an enormous cross-referenced encyclopedia; domain-specific databases about movies, music, sports results, pinball machines, memes, and cocktails; and too many government statistics (some of them nearly true!) from too many governments to wrap your head around.

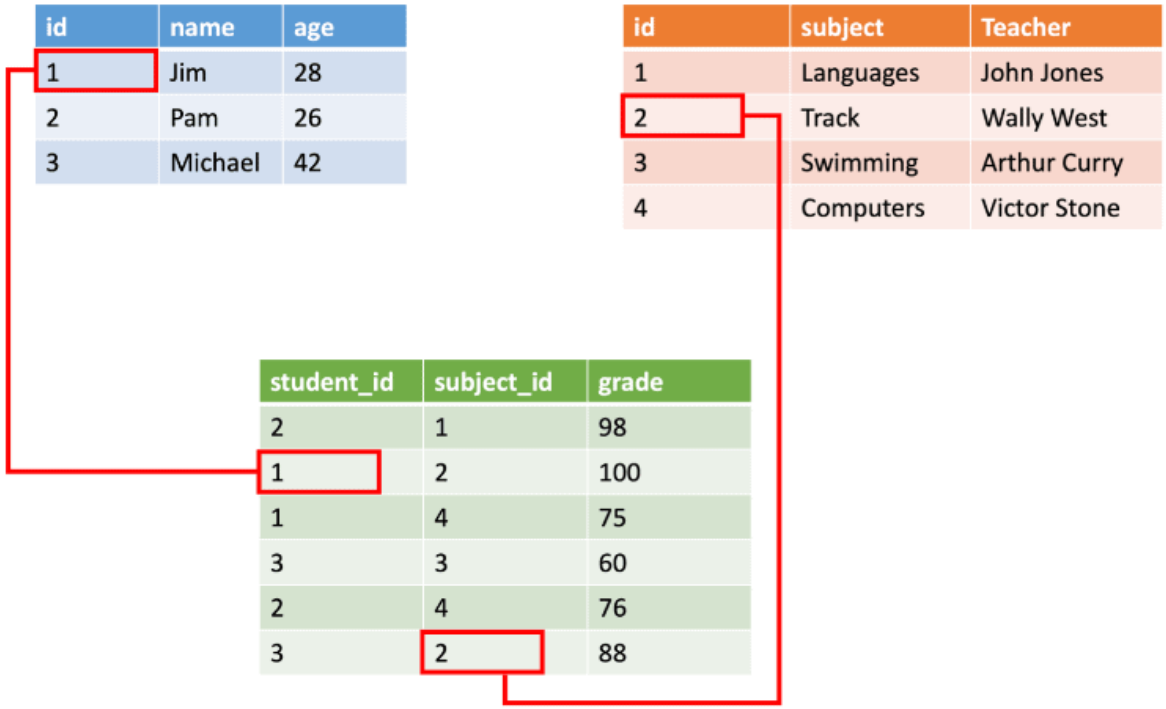
What Is Data Science?

- **What is Data?**
- **Data** is a collection of facts, numbers, measurements, observations. It can be raw or processed and is used to gain insights, make decisions, and drive technologies like AI and machine learning.



What Is Data Science?: Types of Data

- **Structured Data**
  - Organized and stored in a fixed format (e.g., databases, spreadsheets).
  - Examples:
    - Customer records (Name, Age, Email)
    - Sales data (Price, Quantity, Date)
    - SQL Databases



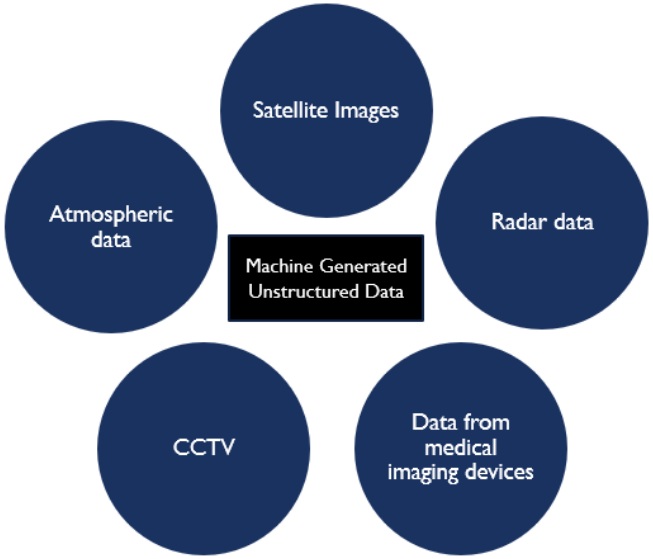
id	name	age
1	Jim	28
2	Pam	26
3	Michael	42

id	subject	Teacher
1	Languages	John Jones
2	Track	Wally West
3	Swimming	Arthur Curry
4	Computers	Victor Stone

student_id	subject_id	grade
2	1	98
1	2	100
1	4	75
3	3	60
2	4	76
3	2	88

What Is Data Science?: Types of Data

- **Unstructured Data**
- No predefined format, making it harder to process.
- Examples:
  - Text documents, emails, social media posts
  - Images, videos, audio files
  - Sensor data, logs



What Is Data Science?: Types of Data

- **Semi-Structured Data**
- Has some organization but doesn't fit traditional databases.
- Examples:
  - JSON, XML files
  - Emails with metadata (subject, timestamp)

Semi-structured data

```
<University>
 <Student ID="1">
  <Name>John</Name>
  <Age>18</Age>
  <Degree>B.Sc.</Degree>
 </Student>
 <Student ID="2">
  <Name>David</Name>
  <Age>31</Age>
  <Degree>Ph.D. </Degree>
 </Student>
....
</University>
```

```
{
  "name": "Jane Smith",
  "email": "jane.smith@example.com",
  "preferences": {
    "newsletter": true,
    "smsNotifications": false
  }
}
```

What Is Data Science?: Sources of Data

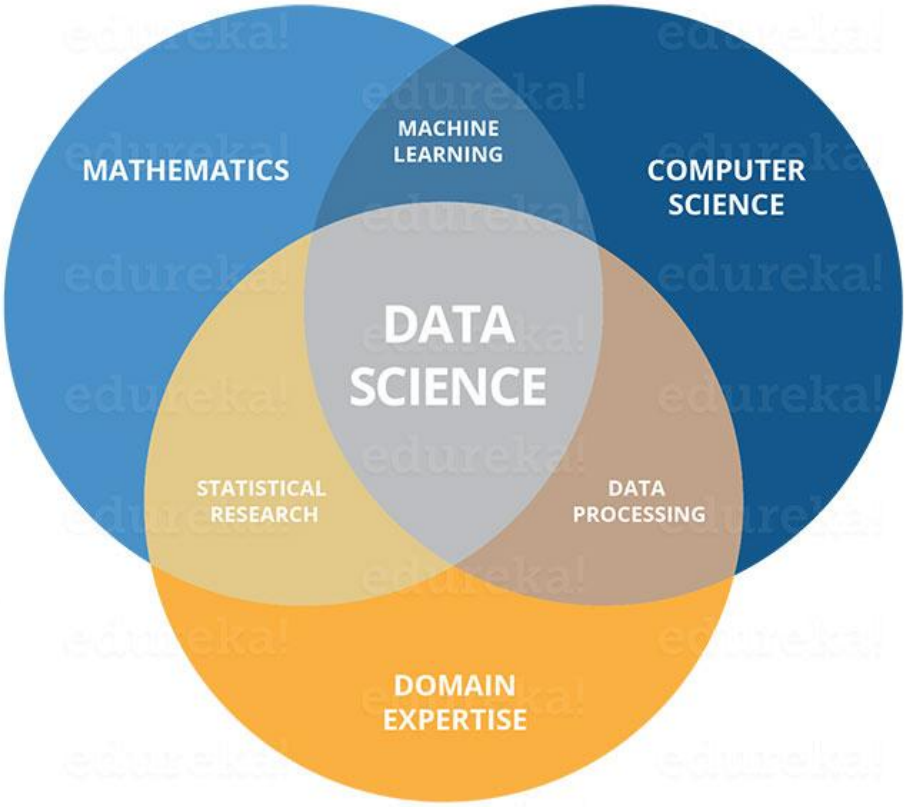
- **Digital Transactions** – Sales records, banking transactions.
- **Social Media** – Posts, likes, shares, and comments.
- **Sensors & IoT Devices** – Temperature sensors, smartwatches.
- **Healthcare Records** – Patient data, medical images.
- **Government & Research** – Census data, scientific experiments.

Why is Data Important?

- **Drives decision-making** – Businesses use data to improve operations.
- **Enables AI & Machine Learning** – Models learn from large datasets.
- **Improves efficiency** – Automates tasks and predicts trends.
- **Personalizes experiences** – Recommender systems (Netflix, Amazon).

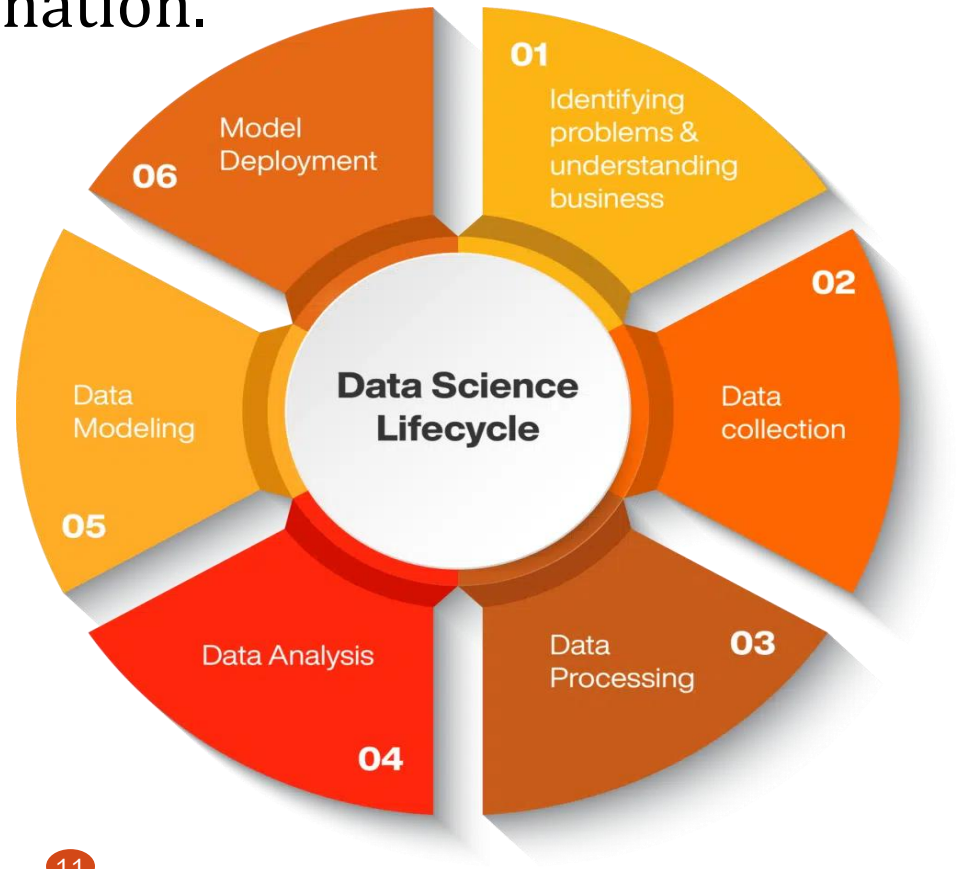
What Is Data Science?

- Data science is the **study** of data to **derive** meaningful insights for businesses. It's a multidisciplinary field that combines **statistics, math, computer engineering, and artificial intelligence** (AI).



Data Science Lifecycle

- This lifecycle encompasses **Six key stages**, that include—Identify problem, data collection, data pre-processing, data analysis, data modeling, and model deployment and dissemination.



Data Science Lifecycle

1. Identifying Problems & Understanding Business

This is the **foundation** of any data science project.

- **Objective:** Understand what the business wants to achieve.
- **Activities:**
  - Meet with stakeholders to gather requirements.
  - Define the problem clearly.
  - Translate business problems into data science problems.
  - Set project goals and success metrics.

Example: A retail company wants to reduce customer churn. The data scientist's job is to understand why customers leave and how to predict it.

2. Data Collection

Once the problem is defined, the next step is to **gather relevant data**.

- **Sources:** Databases, files, APIs, IoT devices, social media, etc.
- **Types:** Structured (tables), unstructured (images, text), or semi-structured (JSON, XML).
- **Goal:** Ensure the data is accurate, relevant, and sufficient for analysis.

Data Science Lifecycle

3. Data Processing

Also known as **Data Cleaning or Data Wrangling**.

- **Objective:** Prepare raw data for analysis.
- **Tasks:**
  - Handle missing or duplicate data.
  - Remove noise and outliers.
  - Convert data types, normalize/scale features.
  - Combine data from multiple sources (integration).

Clean data is **crucial** because poor-quality data leads to inaccurate models.

4. Data Analysis

This is the **exploratory phase** where you try to understand the patterns in the data.

- **Goal:** Gain insights and generate hypotheses.
- **Techniques:**
  - Descriptive statistics (mean, median, mode).
  - Data visualization (charts, graphs, heatmaps).
  - Correlation and trend analysis.

This helps identify **key features** and relationships that will feed into modeling.

Data Science Lifecycle

5. Data Modeling

This is where the **magic** happens – building predictive or prescriptive models.

- **Goal:** Create models that solve the problem.
- **Process:**
  - Choose a machine learning algorithm (e.g., decision trees, logistic regression).
  - Train the model on historical data.
  - Validate using test data or cross-validation.
  - Tune hyperparameters for better performance.

The result is a model that can **predict outcomes** or **classify data** accurately.

6. Model Deployment

Now it's time to **put the model into action** in the real world.

- **Objective:** Make the model accessible to users or systems.
- **Methods:**
  - Integrate into a product (e.g., via an API).
  - Build dashboards or apps for visualization.
  - Automate decision-making processes.

After deployment, the model is **monitored** for accuracy and performance over time. Retraining may be needed as data evolves.

Visualizing Data

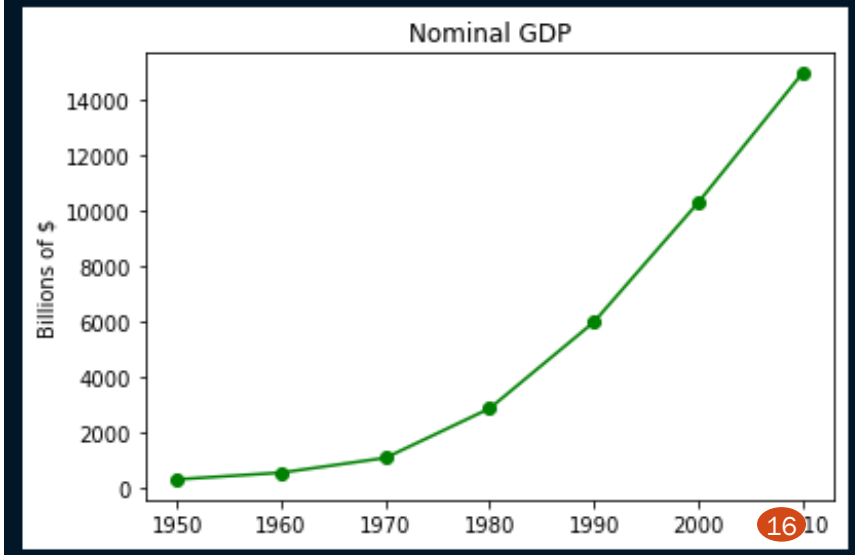
- A fundamental part of the data scientist's toolkit is data visualization.
- Data visualization is essential in today's world because it helps people understand complex data quickly and efficiently.
- Here are some key reasons why it is important:
- **Simplifies Complex Data**
- **Enhances Decision-Making**
- **Identifies Trends and Patterns**
- **Saves Time**

Visualizing Data

Python program to plot Line chart by assuming your own data.

```python
from matplotlib import pyplot as plt
years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]
# create a line chart, years on x-axis, gdp on y-axis
plt.plot(years, gdp, color='green', marker='o', linestyle='solid')
# add a title
plt.title("Nominal GDP")
# add a label to the y-axis
plt.ylabel("Billions of $")
plt.show()

```

✓ 8.7s

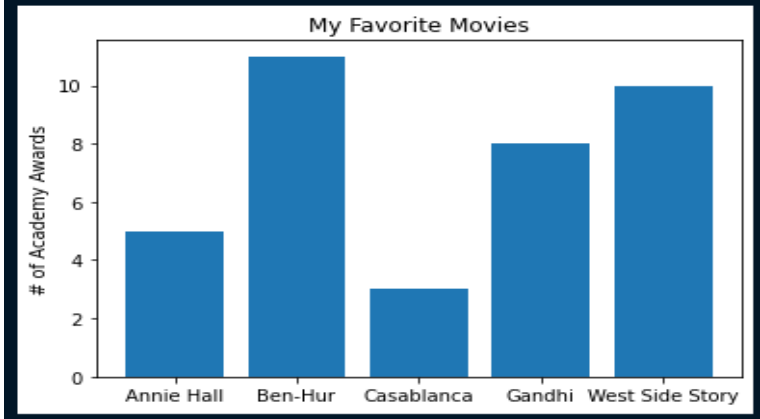


Bar Charts:

- **Python program to plot bar chart by assuming your own data and explain the various attributes of bar chart.**
- A bar chart is a good choice when you want to show how some quantity varies among some discrete set of items.
- ***Syntax: plt.bar(x, height, width, bottom, align)***
- For instance, the below figure shows how many Academy Awards were won by each of a variety of movies.

```
1  from matplotlib import pyplot as plt
2  movies = ["Annie Hall", "Ben-Hur", "Casablanca", "Gandhi", "West Side Story"]
3  num_oscars = [5, 11, 3, 8, 10]
4  # plot bars with left x-coordinates [0, 1, 2, 3, 4], heights [num_oscars]
5  plt.bar(range(len(movies)), num_oscars)
6  plt.title("My Favorite Movies")  # add a title
7  plt.ylabel("# of Academy Awards")  # label the y-axis
8  # label x-axis with movie names at bar centers
9  plt.xticks(range(len(movies)), movies)
10 plt.show()
```

✓ 0.1s



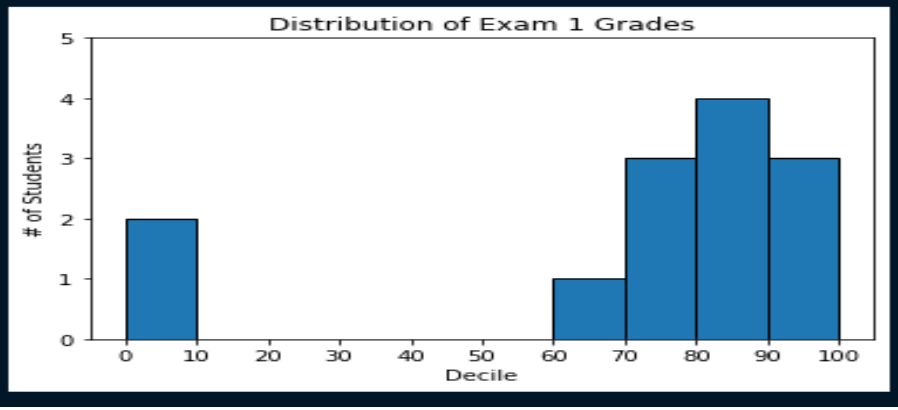
- a

Bar Charts: Histograms

- A bar chart can also be a good choice for plotting histograms of bucketed numeric values, as in the below figure, in order to visually explore how the values are distributed.
- **Python program to plot Histogram by assuming your own data**

```python
from matplotlib import pyplot as plt
from collections import Counter
grades = [83, 95, 91, 87, 70, 0, 85, 82, 100, 67, 73, 77, 0]
# Bucket grades by decile, but put 100 in with the 90s
histogram = Counter(min(grade // 10 * 10, 90) for grade in grades)
plt.bar([x + 5 for x in histogram.keys()],  # Shift bars right by 5
        histogram.values(),  # Give each bar its correct height
        10,  # Give each bar a width of 10
        edgecolor=(0, 0, 0))  # Black edges for each bar
plt.axis([-5, 105, 0, 5])  # x-axis from -5 to 105,
# y-axis from 0 to 5
plt.xticks([10 * i for i in range(11)])  # x-axis labels at 0, 10, ...,100
plt.xlabel("Decile")
plt.ylabel("# of Students")
plt.title("Distribution of Exam 1 Grades")
plt.show()
```

0.1s



Bar Charts

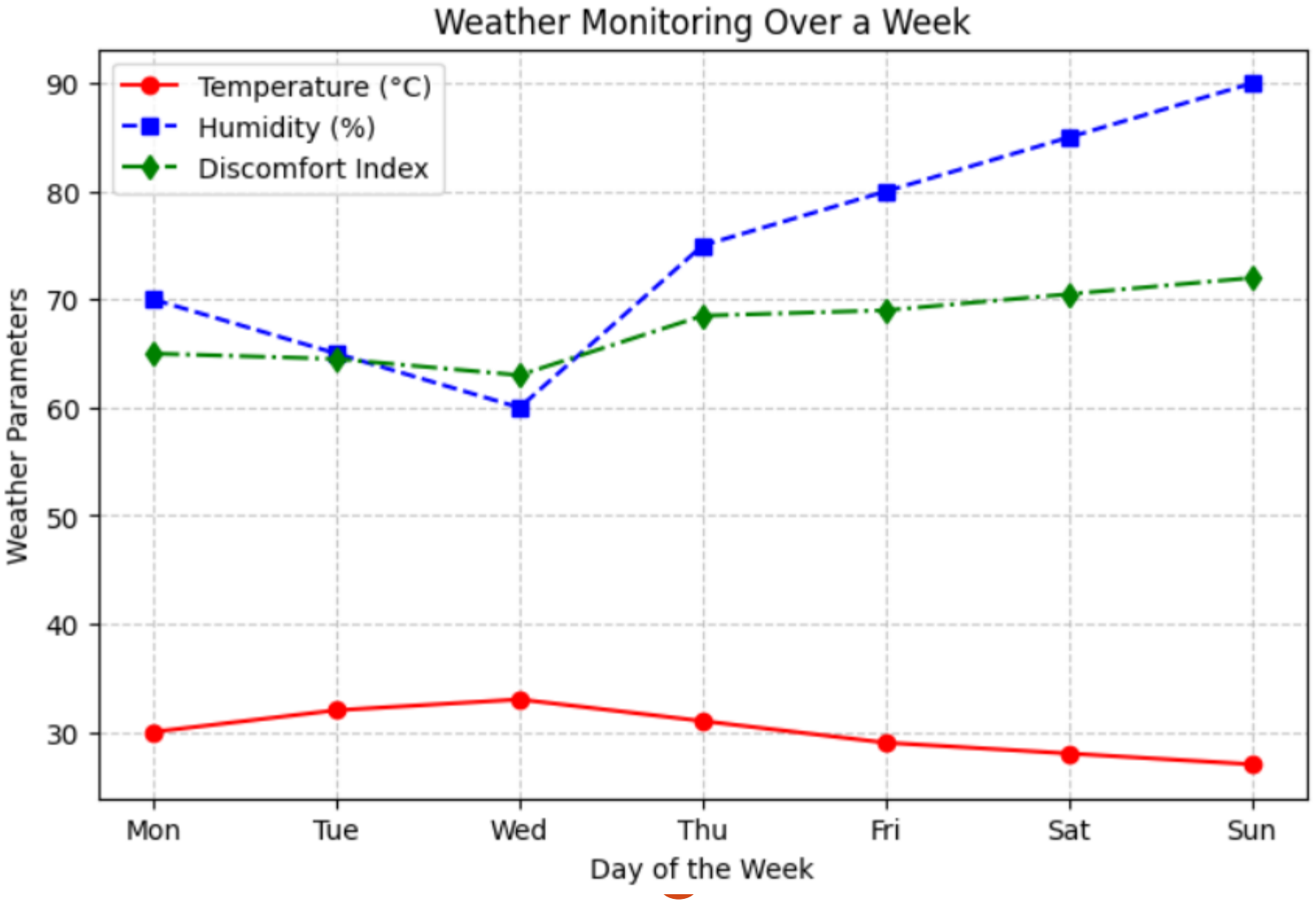
- The third argument to ***plt.bar*** specifies the bar width. Here we chose a width of 10, to fill the entire decile.
- We also shifted the bars right by 5, so that, for example, the "10" bar (which corresponds to the decile 10–20) would have its center at 15 and hence occupy the correct range.
- We also added a black edge to each bar to make them visually distinct.
- The call to ***plt.axis*** indicates that we want the x-axis to range from –5 to 105 (just to leave a little space on the left and right), and that the y-axis should range from 0 to 5.
- And the call to ***plt.xticks*** puts x-axis labels at 0, 10, 20, …, 100.

Multiple Line Charts: using plot()

Python program to plot Multiple Line Charts by assuming your own data

```
import matplotlib.pyplot as plt
# Sample data: Days of the week
days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
temperature = [30, 32, 33, 31, 29, 28, 27] # Temperature (°C)
humidity = [70, 65, 60, 75, 80, 85, 90] # Humidity (%)
# Discomfort Index (A rough measure of heat discomfort:Temp + 0.5 * Humidity)
discomfort_index = [t + 0.5 * h for t, h in zip(temperature, humidity)]
plt.figure(figsize=(8, 5)) # Create the line plot
plt.plot(days, temperature, 'r-o', label="Temperature (°C)")  # Red solid line with circles
plt.plot(days, humidity, 'b--s', label="Humidity (%)")  # Blue dashed line with squares
plt.plot(days, discomfort_index, 'g-.d', label="Discomfort Index")  # Green dot-dashed line with diamonds
plt.xlabel("Day of the Week")
plt.ylabel("Weather Parameters")
plt.title("Weather Monitoring Over a Week")
plt.legend() # Adding legend
plt.grid(True, linestyle="--",) # Grid for better readability
```

Multiple Line Charts

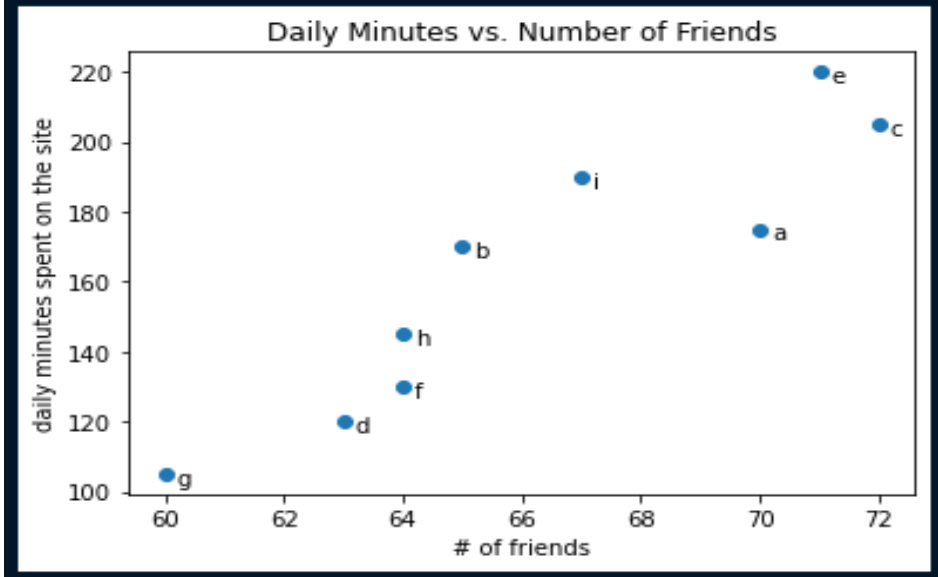


Scatterplots

- A scatterplot is the right choice for visualizing the relationship between two paired sets of data.
- For example, the below figure illustrates the relationship between the number of friends your users have and the number of minutes they spend on the site every day:
- **Python program to plot scatterplots by assuming your own data**

```python
from matplotlib import pyplot as plt
friends = [70, 65, 72, 63, 71, 64, 60, 64, 67]
minutes = [175, 170, 205, 120, 220, 130, 105, 145, 190]
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
plt.scatter(friends, minutes)
# label each point
for label, friend_count, minute_count in zip(labels, friends, minutes):
    plt.annotate(label,
                 xy=(friend_count, minute_count),  # Put the label with its point
                 xytext=(5, -5),  # but slightly offset
                 textcoords='offset points')
plt.title("Daily Minutes vs. Number of Friends")
plt.xlabel("# of friends")
plt.ylabel("daily minutes spent on the site")
plt.show()

```

✓ 0.1s



Linear Algebra: Vectors

- Linear algebra is the branch of mathematics that deals with vector spaces.

Physical quantity with magnitude and no direction is known as a **scalar** quantity and a physical quantity with magnitude and directions is known as a **vector** quantity.

Vectors are objects that can be added together to form new vectors and that can be multiplied by scalars (i.e., numbers), also to form new vectors

Vectors, are often a useful way to represent numeric data.

For example, if you have the heights, weights, and ages of a large number of people, you can treat your data as **three-dimensional vectors** [height, weight, age].

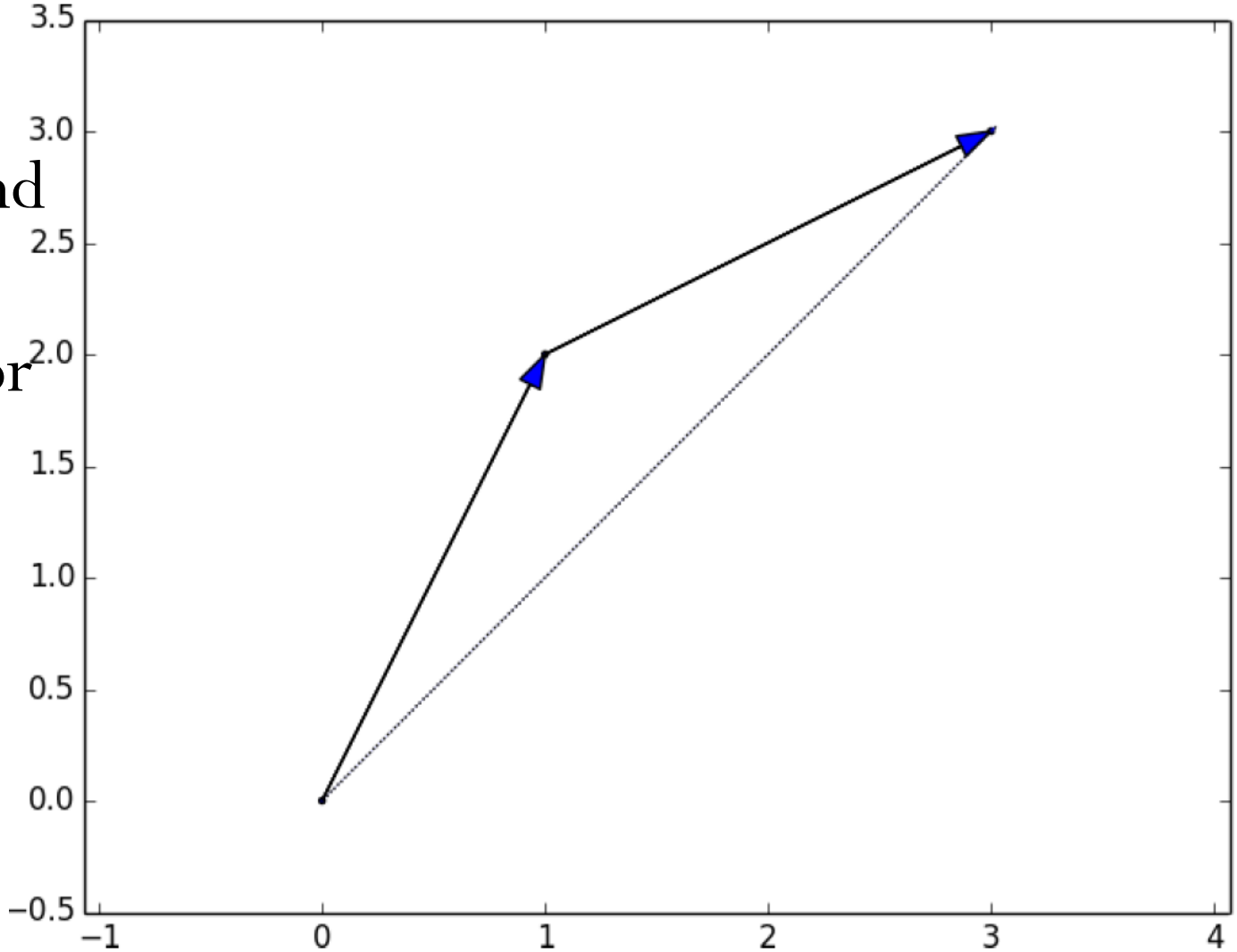
```
height_weight_age = [70, # inches,
                     170, # pounds,
                     40 ] # years
```

Vectors

- **Perform arithmetic on vectors.**
- Because Python lists aren't vectors (and hence provide no facilities for vector arithmetic), we'll need to build these arithmetic tools ourselves.
- Add two vectors. Vectors add component wise.
- This means that if two vectors **v and w** are the same length, their sum is just the vector whose first element is v[0] + w[0], whose second element is v[1] + w[1], and so on.
- (If they're not the same length, then we're not allowed to add them.)

Adding two Vectors

For example, adding the vectors [1, 2] and [2, 1] results in [1 + 2, 2 + 1] or [3, 3], as shown in Figure 4-1.



Add two Vectors

```
def vector_add(v, w):
  """adds corresponding elements"""
  return [v_i + w_i   for v_i, w_i in zip(v, w)]
print(vector_add([5, 7, 9], [4, 5, 6]))
```

To subtract two vectors we just subtract the corresponding elements:

```
def vector_subtract(v, w):
  """subtracts corresponding elements"""
  return [v_i - w_i   for v_i, w_i in zip(v, w)]
print(vector_subtract([5, 7, 9], [4, 5, 6]))
```

Component wise sum a list of Vectors

- We'll also sometimes want to component wise sum a list of vectors—that is, create a new vector whose first element is the sum of all the first elements, whose second element is the sum of all the second elements, and so on:

```
def vector_sum(vectors):
    """sums all corresponding elements"""
    result = vectors[0] # start with the first vector
    for vector in vectors[1:]: #loop over the others
        result = vector_add(result, vector) #add to result
        return result
print(vector_sum([[1, 2], [3, 4], [5, 6], [7, 8]]))
```

Multiply a vector by a scalar

- We'll also need to be able to multiply a vector by a scalar, which we do simply by multiplying each element of the vector by that number:

```
def scalar_multiply(c, v):
    """c is a number, v is a vector"""
    return [c * v_i for v_i in v]
print(scalar_multiply(2, [1, 2, 3]))
```

Compute Component wise Means

```
def vector_sum(vectors):
  """sums all corresponding elements"""
  result = vectors[0] # start with the first vector
  for vector in vectors[1:]: #loop over the others
    result = vector_add(result, vector) #add to result
    return result
def scalar_multiply(c, v):
  """c is a number, v is a vector"""
  return [c * v_i for v_i in v]
def vector_mean(vectors):
  """compute the vector whose ith element is the mean of the
  ith elements of the input vectors"""
  n = len(vectors)
  return scalar_multiply(1/n, vector_sum(vectors))

print(vector_mean([[1, 2], [3, 4], [5, 6]]))
```

[1.3333333333333333, 2.0]

Dot Product

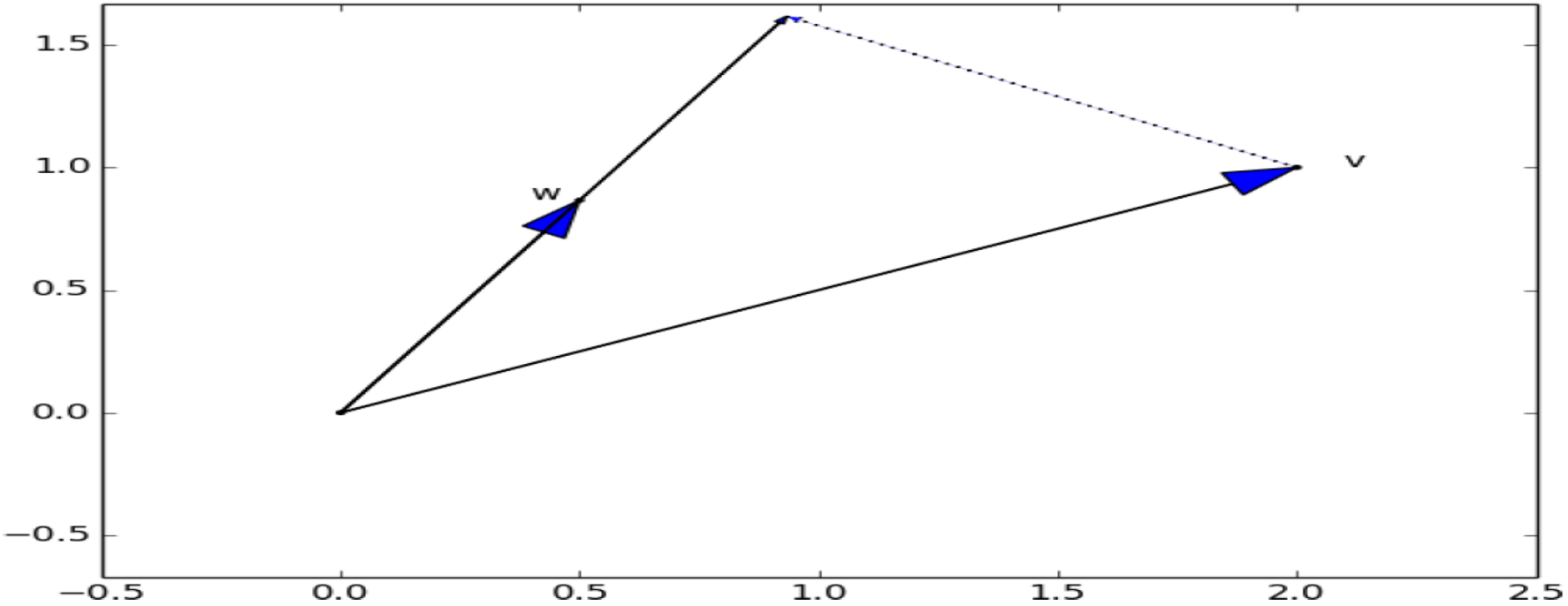
- The dot product of two vectors is the sum of their component wise products:

```
def dot(v, w):
  """v_1 * w_1 + ... + v_n * w_n"""
  return sum(v_i * w_i for v_i, w_i in zip(v, w))
print(dot([1, 2, 3], [4, 5, 6]))
```

```
32
```

Dot Product

- If w has magnitude 1, the dot product measures how far the vector v extends in the w direction.
- For example, if ***w = [1, 0]***, then ***dot(v, w)*** is just the first component of v.
- Another way of saying this is that it's the length of the vector you'd get if you projected ***v*** onto ***w***.



The dot product as vector projection

Sum of Squares

- Using this, it's easy to compute a vector's sum of squares:

```
def sum_of_squares(v):
  """v_1 * v_1 + ... + v_n * v_n"""
  return dot(v, v)
print(sum_of_squares([1, 2, 3]))
```

Compute magnitude (or length)

Python program to compute the magnitude of vectors

```
def dot(v, w):
  """v_1 * w_1 + ... + v_n * w_n"""
  return sum(v_i * w_i for v_i, w_i in zip(v,w))

def sum_of_squares(v):
  """v_1 * v_1 + ... + v_n * v_n"""
  return dot(v, v)
```

```
import math
def magnitude(v):
  return math.sqrt(sum_of_squares(v))
# math.sqrt is square root function
print(magnitude([1,2,3]))
```

Compute Distance Between Two Vectors

- To compute the distance between two vectors, defined as:

$$\sqrt{(v_1 - w_1)^2 + \ldots + (v_n - w_n)^2}$$

Python program to compute the Distance between of vectors

```
def vector_subtract(v, w):
  """subtracts corresponding elements"""
  return [v_i - w_i   for v_i, w_i in zip(v, w)]

def squared_distance(v, w):
  """(v_1 - w_1) ** 2 + ... + (v_n - w_n) ** 2"""
 return sum_of_squares(subtract(v, w))

def distance(v, w):
  return math.sqrt(squared_distance(v, w))
print(distance([3,4],[1,2]))
```

Matrices

- A matrix is a two-dimensional collection of numbers.
- We will represent matrices as lists of lists, with each inner list having the same size and representing a row of the matrix.
- If ***A*** is a matrix, then ***A[i][j]*** is the element in the ***ith*** row and the ***jth*** column. Per mathematical convention, we will frequently use capital letters to represent matrices. For example:

Vector using List and assert

```
from typing import List
Vector=List[float]
def add(v: Vector, w: Vector) -> Vector:
  """Adds corresponding elements"""
   assert len(v) == len(w), "vectors must be the same length"
  return [v_i+w_i for v_i, w_i in zip(v, w)]
print(add ([5, 7, 9], [4, 5, 6]))
```

Matrices

- For example:

```
# Another type alias
Matrix = List[List[float]]

A = [[1, 2, 3],  # A has 2 rows and 3 columns
     [4, 5, 6]]

B = [[1, 2],     # B has 3 rows and 2 columns
     [3, 4],
     [5, 6]]
```

Get Row

- If a matrix has n rows and k columns, we will refer to it as an n × k matrix.
- We can (and sometimes will) think of each row of an n × k matrix as a vector of length k, and each column as a vector of length n:

```
# Get a row from matrix
from typing import List

Vector= List[float]
Matrix = List[List[float]]
def get_row(A: Matrix , i:int)->Vector:
  """ Return ith row"""
  return A[i]
print("Display the first row of matrix")
print(get_row([[1,2,3],[4,5,6]],0))
print("Display the second row of matrix")
print(get_row([[1,2,3],[4,5,6]],1))
```

```
Display the first row of matrix
[1, 2, 3]
Display the second row of matrix
[4, 5, 6]
```

Get Column

```
#Get Column from a Matrix
from typing import List

Vector=List[float]
Matrix = List[List[float]]

def get_col(A:Matrix,j:int) -> Vector:
  return [A_i[j] for A_i in A]

print("Display First Column of Matrix")
print(get_col([[1,2,3],[4,5,6]],0))

print("Display First Column of Matrix")
print(get_col([[1,2,3],[4,5,6]],1))

print("Display First Column of Matrix")
print(get_col([[1,2,3],[4,5,6]],2))
```

```
Display First Column of Matrix
[1, 4]
Display First Column of Matrix
[2, 5]
Display First Column of Matrix
[3, 6]
```

Matrix rows and columns

- Given this list-of-lists representation, the matrix ***A*** has ***len(A)*** rows and ***len(A[0])*** columns, which we consider its shape:

```
# Get shape of a matrix

from typing import Tuple

def shape(A:Matrix) -> Tuple:
  n_row=len(A)
  n_col=len(A[0])
  return n_row,n_col

print(shape([[1,2,3],[4,5,6],[8,9,10]]))
```

```
(3,3)
```

Lambda Functions

```
1  # Here is an example of lambda function that doubles the input value.
2
3  # Program to show the use of lambda functions
4  double = lambda x: x * 2
5
6  print(double(5))
7
8  # In the above program, lambda x: x * 2 is the lambda function.
9  # Here x is the argument and x * 2 is the expression that gets evaluated and returned.
```

✓ 3.9s

```
10
```

Create Matrix

- We'll also want to be able to create a matrix given its shape and a function for generating its elements.
- We can do this using a nested list comprehension

```
from typing import List
from typing import Callable

Vector=List[float]
Matrix = List[List[float]]

def make_matrix(num_rows, num_cols, entry_fn):
"""returns a num_rows num_cols,(i,j)th entry is entry_fn(i, j)"""
  return [[entry_fn(i, j) # given i, create a list
  for j in range(num_cols)] # [entry_fn(i, 0), ... ]
  for i in range(num_rows)] # create one list for each i
print(make_matrix(2,3, lambda i,j:i*j))
```

```
[[0, 0, 0], [0, 1, 2]]
```

Identity Matrix

- you could make a 5 × 5 identity matrix (with 1s on the diagonal and 0s elsewhere) like so:

```
def is_diagonal(i, j):
  """1's on the 'diagonal', 0's everywhere else"""
  return 1 if i == j else 0
identity_matrix = make_matrix(5, 5, is_diagonal)
print(identity_matrix)
```

Statistics

- Statistics refers to the mathematics and techniques with which we understand data.
- Required for large data set .
- Imagine staring at a list of 1 million numbers.
- For that reason we use statistics to distill and communicate relevant features of our data.

Describing a Single Set of Data

- One obvious description of any dataset is simply the data itself:

```
num_friends = [100, 49, 41, 40, 25,
               # ... and lots more
              ]
```

- For a small enough dataset, this might even be the best description.
- But for a larger dataset, this is unwieldy and probably opaque. (Imagine staring at a list of 1 million numbers.)
- For that reason, we use statistics to distill and communicate relevant features of our data.

Central Tendencies → Mean

- Central tendency is a statistical concept in data science that describes the center or typical value of a dataset. **It helps summarize a large set of data points by identifying a single representative value.**
- Usually, we'll want some notion of where our data is centered.
- Most commonly we'll use the mean (or average), which is just the sum of the data divided by its count:
- **Python program to compute the Central Tendencies(Mean, Median, Mode) of vectors**
- **Python program to compute the mean of vectors**

```
num_friends=[100,49,41,40,25,21,21,19,19,18,18,16,15,6,6,6,6,6,6
             ,6,6,6,6,6,6,6,5,5,5]
def mean(x):
   return sum(x) / len(x)
mean(num_friends)
```

17.06896551724138

Median

- We'll also sometimes be interested in the median, which is the **middlemost value** (if the number of data points is odd) or the aver**age of the two middle-most values** (if the number of data points is even).
- For instance, if we have five data points in a **sorted vector x**, the median is x[5 // 2] or x[2].
- If we have six data points, we want the average of x[2] (the third point) and x[3] (the fourth point).
- Notice that—unlike the mean—the **median doesn't fully depend on every value in your data.**
- For example, if you make the largest point larger (or the smallest point smaller), the middle points remain unchanged, which means so does the median.
- The mean is skewed due to an extreme outlier.
- The median provides a fairer estimate

Median

Python program to compute the median of vectors

```
num_friends=[100,49,41,40,25,21,21,19,19,18,18,16,15,6,6,6,6,
6,6,6,6,6,6,6,6,6,5,5,5]
def median(v):
  """finds the 'middle-most' value of v"""
  n = len(v)
  sorted_v = sorted(v)
  midpoint = n // 2
  if n % 2 == 1:
    # if odd, return the middle value
    return sorted_v[midpoint]
  else:
# if even, return the average of the middle values
    lo = midpoint - 1
    hi = midpoint
    return (sorted_v[lo] + sorted_v[hi]) / 2
median(num_friends) # 6.0
```

Percentiles, Quartiles and Quantiles

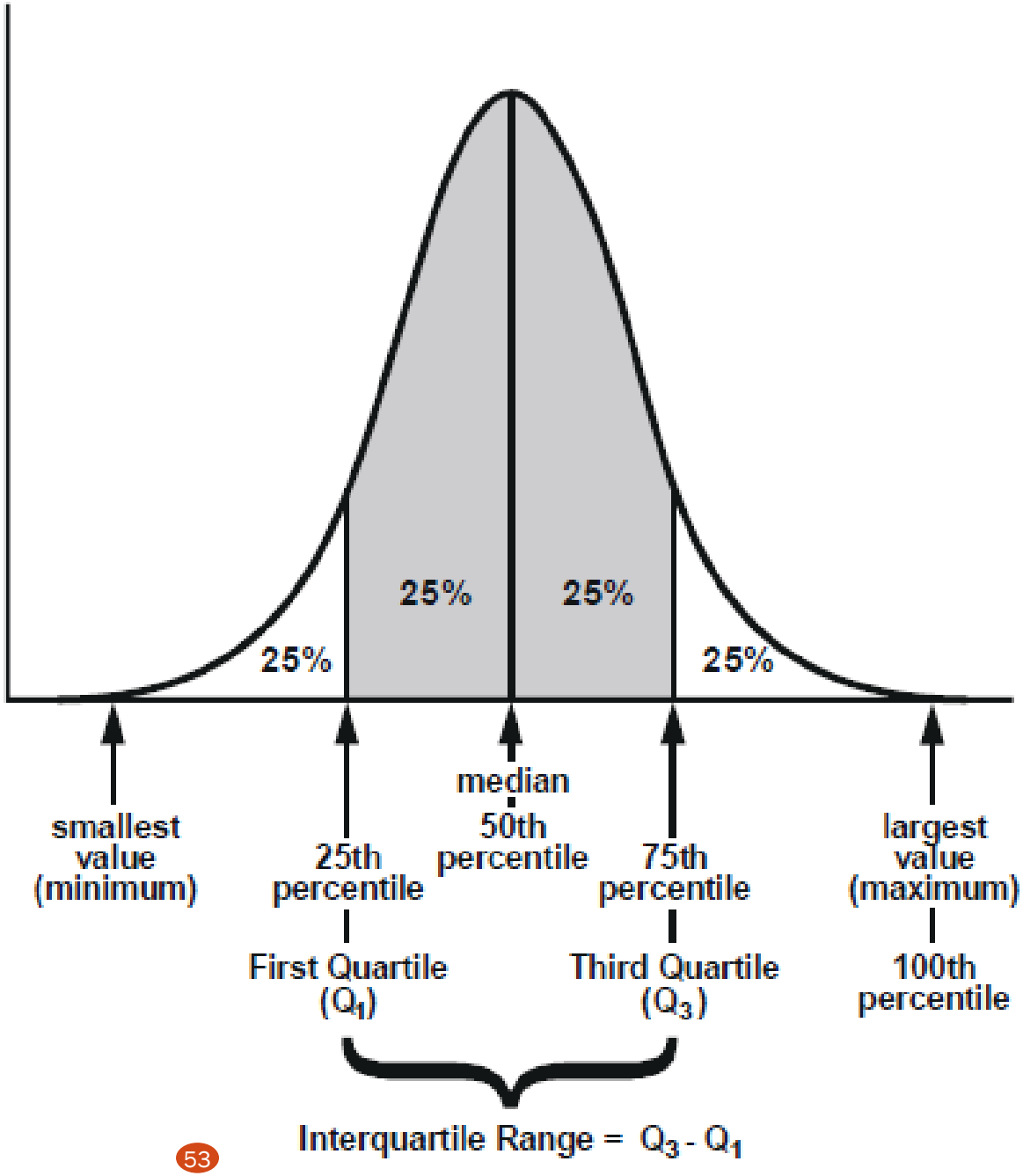
- In **statistics**, **percentiles, quartiles, and quantiles** are used to divide data into different parts to **analyze distributions effectively**. These terms are closely related but differ in the number of divisions they create. Here's a simple definition of each:
- ***Percentiles:*** Range from 0 to 100.
  - **Use Case**: Used in test scores, income distribution, and healthcare analysis.
- ***Quartiles:*** Range from 0 to 4.
- Breakdown:
  - Q1 (25th percentile): The value below which 25% of data falls.
  - Q2 (50th percentile or Median): The middle value of the dataset.
  - Q3 (75th percentile): The value below which 75% of data falls.
  - Q4 (100th percentile): The maximum value in the dataset.
  - Use Case: Used in box plots, income distribution, and performance analysis.

Percentiles, Quartiles and Quantiles

- ***Quantiles:*** Range from any value to any other value.
- Percentiles and quartiles are types of quantiles.
- Some types of quantiles even have specific names, including:
- 4-quantiles are called ***quartiles***.
- 5-quantiles are called ***quintiles***.
- 8-quantiles are called ***octiles***.
- 10-quantiles are called ***deciles***.
- 100-quantiles are called ***percentiles***.

Median

This diagram is called a Box Plot Distribution or Quartile Distribution. It represents the interquartile range (IQR) and the percentile breakdown of a dataset in a normal distribution curve.



Quantile

- A generalization of the median is the quantile, which represents the value under which a certain percentile of the data lies (the median represents the value under which 50% of the data lies):

Python program to compute quantile of vectors

```
num_friends = [70,60,66,49,41,40,25,21,21,19,19,18,18,16,15,6,6,6,6,6,6,6,6,6,6,6,6,5,5,5,5]
print(sorted(num_friends))
print("length of list:")
print(len(num_friends))
def quantile(x, p):
  """returns the pth-percentile value in x"""
  p_index = int(p * len(x))
  print("p_index=", p_index)
  return sorted(x)[p_index]
print("10% of Quantile is",quantile(num_friends, 0.10))
print("25% (Q1) Quantile is",quantile(num_friends, 0.25))
print("75% (Q3) Quantile is", quantile(num_friends, 0.75))
print("90% of Quantile is",quantile(num_friends, 0.90))
```

Mode

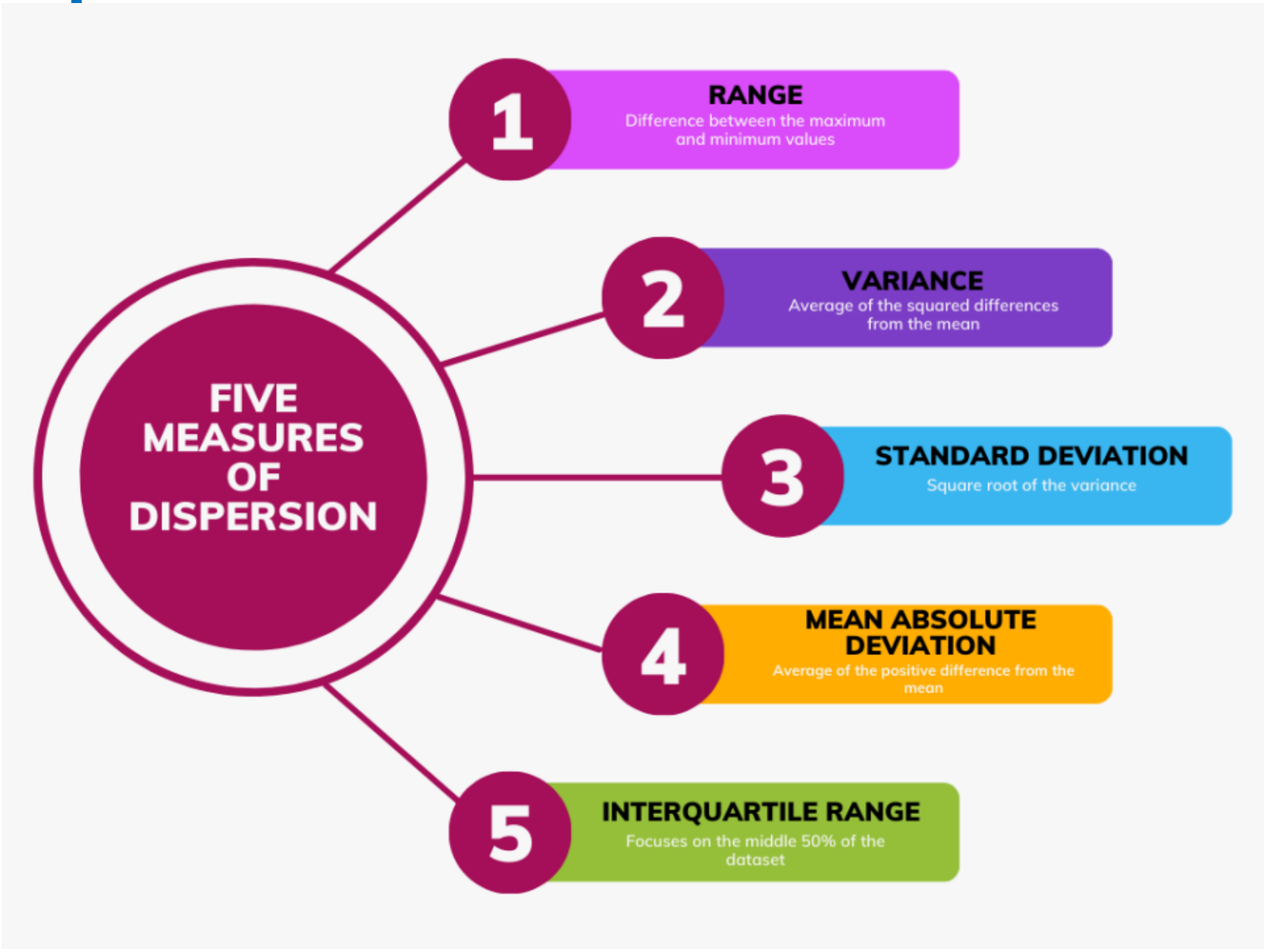
- Mode is defined as a value that occurs most frequently in a dataset.
- For Example, In {6, 9, 3, 6, 6, 5, 2, 3}, the Mode is 6 as it occurs most often.
- The Mode of data set A = {100, 80, 80, 95, 95, 100, 90, 90,100 ,95 } is 80, 90, 95, and 100 because all the four values are repeated twice in the given set.
- The Mode of data set B = {11, 12, 12, 14, 15, 16, 16, 16, 16, 18, 20, 20, 24, 26}
- Here, we get 16 four times, 12 and 20 twice each, and other terms only once. Hence, the Mode for a given set of data is 16.

Mode

Python program to compute the mode of vectors

```
from collections import Counter
num_friends =
[100,49,41,40,25,21,21,19,19,18,18,16,15,6,6,6,
5,5,5]

def mode(x):
    """returns a list, might be more than one
mode"""
    counts = Counter(x)
    max_count = max(counts.values())
    return [x_i for x_i, count in counts.items() if count == max_count]
mode(num_friends) # 1 and 6
```

Dispersion


FIVE MEASURES OF DISPERSION
1
RANGE
Difference between the maximum and minimum values
2
VARIANCE
Average of the squared differences from the mean
3
STANDARD DEVIATION
Square root of the variance
4
MEAN ABSOLUTE DEVIATION
Average of the positive difference from the mean
5
INTERQUARTILE RANGE
Focuses on the middle 50% of the dataset

Dispersion

Key Measures of Dispersion

1. **Range**
   - Difference between the maximum and minimum values.
   - Formula:

$$\text{Range} = \max(X) - \min(X)$$

   - **Example:** If dataset = {2, 5, 8, 10}, then **Range = 10 - 2 = 8.**
2. **Variance (σ^2)**
   - Measures the average squared deviation from the mean.
   - Formula (for population variance):

$$\sigma^2 = \frac{\sum(X_i - \mu)^2}{N}$$

   - **Example:** If dataset = {2, 4, 6}, Mean (μ) = 4,

$$\sigma^2 = \frac{(2-4)^2 + (4-4)^2 + (6-4)^2}{3} = \frac{4+0+4}{3} = 2.67$$

Dispersion

3. **Standard Deviation (σ or s)**
   - Square root of variance.
   - Formula:

$$\sigma = \sqrt{\sigma^2}$$

   - It provides dispersion in the same unit as the data.
4. **Interquartile Range (IQR)**
   - Measures the spread of the middle 50% of the data.
   - Formula:

$$IQR = Q3 - Q1$$

   - **Example:** If Q1 = 25 and Q3 = 75, then **IQR = 75 - 25 = 50.**

Dispersion

- Dispersion refers to measures of how spread out our data is.
- Typically they're statistics for which values near zero signify not spread out at all and large values (whatever that means) signify very spread out.
- For instance, a very simple measure is the **range**, which is just the difference between the largest and smallest elements:

```
# "range" already means something in Python, so we'll use a
different name
num_friends=[100,49,41,40,25,21,21,19,19,18,18,16,15,6,6,6,5,5,5]
def data_range(x):
  return max(x) - min(x)
data_range(num_friends) # 99
```

Variance

- The term variance refers to a statistical measurement of the spread between numbers in a data set.
- More specifically, variance measures how far each number in the set is from the mean and thus from every other number in the set.
- Variance is often depicted by this symbol: $\boldsymbol{\sigma^2}$.

Understanding Variance

- In statistics, variance measures variability from the average or mean.
- It is calculated by taking the differences between each number in the data set and the mean, then squaring the differences to make them positive, and finally dividing the sum of the squares by the number of values in the data set.
- Variance is calculated by using the following formula:

$$\text{variance } \sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}$$

where:

$x_i = i^{th}$ data point

$\bar{x} =$ Mean of all data points

$n =$ Number of data points

- The use of n − 1 instead of n in the formula for the sample variance is known as Bessel's correction, which corrects the bias in the estimation of the population variance

Variance

Python program to compute variance of vectors

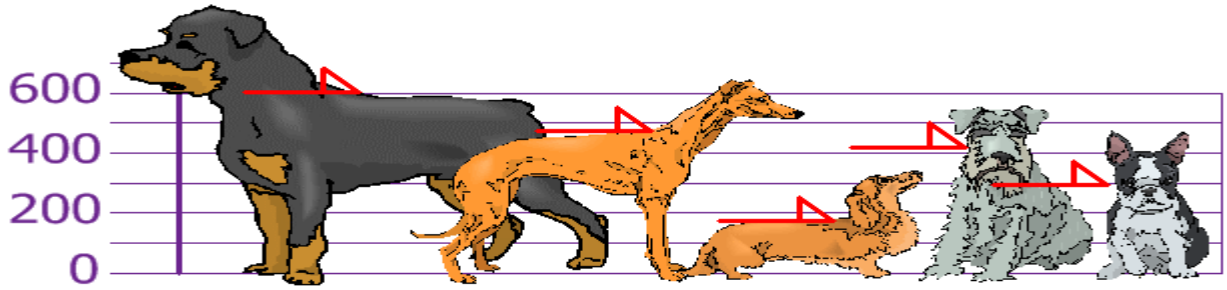
```
# Given dataset
num_friends = [100, 49, 41, 40, 25, 21, 21, 19, 19, 18, 18, 16,
15, 6, 6, 6, 5, 5, 5]

# Step 1: Calculate the mean
mean_value = sum(num_friends) / len(num_friends)

# Step 2: Compute squared differences from the mean
squared_diffs = [(x - mean_value) ** 2 for x in num_friends]

# Step 3: Calculate variance
population_variance = sum(squared_diffs) / len(num_friends)  #
Population variance
sample_variance = sum(squared_diffs) / (len(num_friends) - 1)  #
Sample variance (ddof=1)

# Print results
print(f"Population Variance: {population_variance:.2f}")
print(f"Sample Variance: {sample_variance:.2f}")
```

Standard Deviation

- The Standard Deviation is a measure of how spread out numbers are.
- Its symbol is σ (the greek letter sigma)
- The formula is easy: it is the square root of the Variance. So now you ask, "What is the Variance?

Standard Deviation

- Example
- Measure the heights of dogs (in millimeters):



- The heights (at the shoulders) are: 600mm, 470mm, 170mm, 430mm and 300mm.
- Find out the Mean, the Variance, and the Standard Deviation.
- Your first step is to find the Mean:

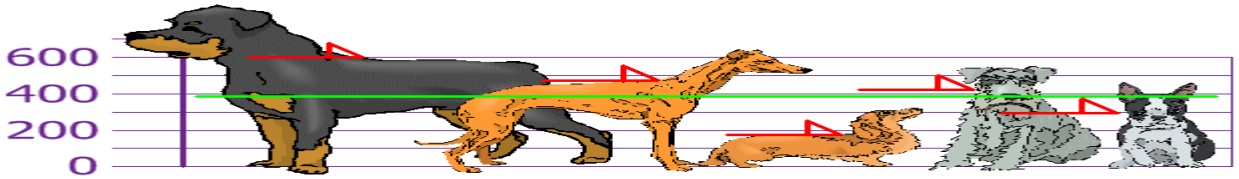
$$\text{Mean} = \frac{600 + 470 + 170 + 430 + 300}{5}$$

- $$= \frac{1970}{5}$$

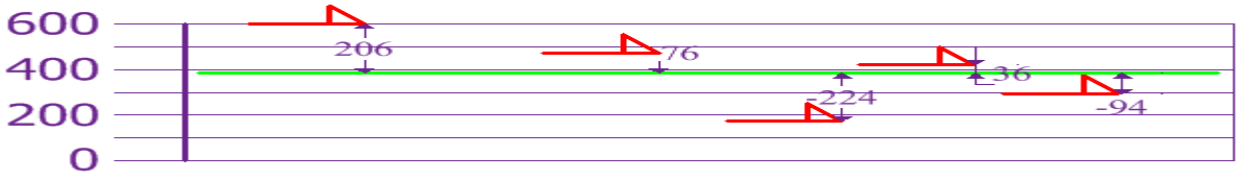
$$= 394$$

Standard Deviation

- so the mean (average) height is 394 mm. Let's plot this on the chart:



- Now we calculate each dog's difference from the Mean:



- To calculate the Variance, take each difference, square it, and then average the result:

Variance

$$\sigma^2 = \frac{206^2 + 76^2 + (-224)^2 + 36^2 + (-94)^2}{5}$$

$$= \frac{42436 + 5776 + 50176 + 1296 + 8836}{5}$$

$$= \frac{108520}{5}$$

$$= 21704$$

- So the Variance is 21,704

Standard Deviation

- And the Standard Deviation is just the square root of Variance, so:

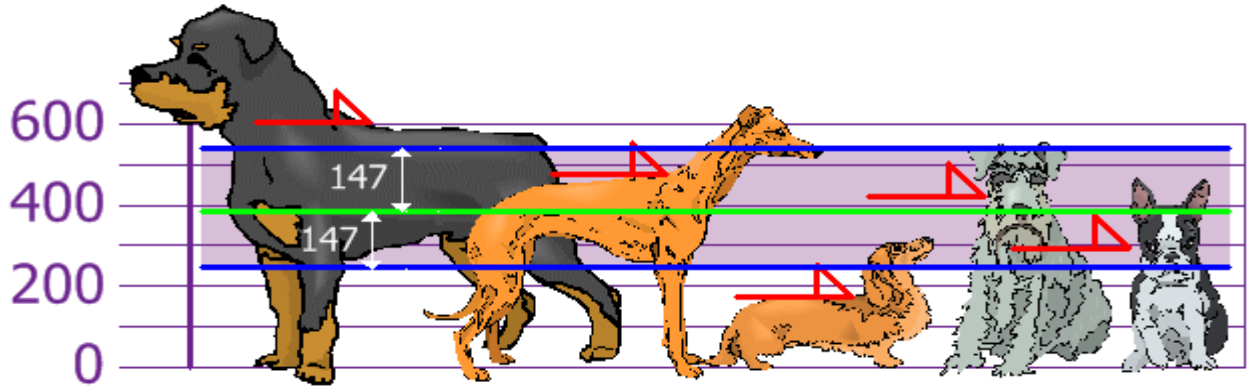
Standard Deviation

$\sigma = \sqrt{21704}$

$= 147.32...$

$= \mathbf{147}$ *(to the nearest mm)*

- Show which heights are within one Standard Deviation (147mm) of the Mean:



- So, using the Standard Deviation we have a "standard" way of knowing what is normal, and what is extra large or extra small.
- Rottweilers are tall dogs. And Dachshunds are a bit short, right?

Standard Deviation

Python program to compute standard deviation of vectors

```
num_friends = [100, 49, 41, 40, 25, 21, 21, 19, 19, 18, 18, 16, 15, 6, 6, 6, 5, 5, 5]
# Step 1: Calculate the mean
mean_value = sum(num_friends) / len(num_friends)
# Step 2: Compute squared differences from the mean
squared_diffs = [(x - mean_value) ** 2 for x in num_friends]
# Step 3: Calculate variance
population_variance = sum(squared_diffs) / len(num_friends)   # Population variance
sample_variance = sum(squared_diffs) / (len(num_friends) - 1)
print(f"Population Variance: {population_variance:.2f}")
print(f"Sample Variance: {sample_variance:.2f}")
std_var=math.sqrt(sample_variance)
print("Standard Deviation",std_var)
```

Difference between Percentile Values

- A more robust alternative computes the difference between the 75th percentile value and the 25th percentile value:

```
num_friends = [100, 49, 41, 40, 25, 21, 21, 19,
19, 18, 18, 16, 15, 6, 6, 6, 5, 5, 5]
def interquartile_range(x):
   return quantile(x, 0.75) - quantile(x, 0.25)
print(interquartile_range(num_friends))
```

- which is quite plainly unaffected by a small number of outliers.

Correlation and Covariance

- Both **correlation** and **covariance** measure the relationship between two variables.

Covariance

- Variance measures how a **single variable** deviates from its mean, covariance measures how **two variables** vary from their means.
- Covariance formula is a statistical formula which is used to assess the **relationship between two variables**.
- It helps to know whether the two variables vary together or change together.
- The covariance is denoted as Cov(X,Y) and the formulas for covariance are given below.

x_i = data value of x

y_i = data value of y

$\bar{x}$ = mean of x

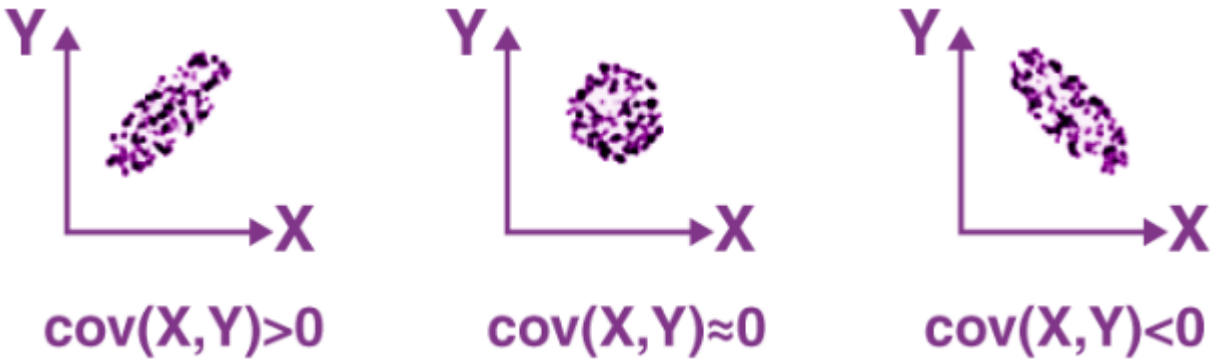
$\bar{y}$ = mean of y

N = number of data values.

$$Cov(x,y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N - 1}$$

Covariance of X and Y

- Below figure shows the covariance of X and Y.



- If ***cov(X, Y)*** is greater than zero, then we can say that the covariance for any two variables is **positive** and both the variables move in the **same direction**.
- If ***cov(X, Y)*** is less than zero, then we can say that the covariance for any two variables is **negative** and both the variables move in the **opposite direction**.
- If ***cov(X, Y)*** is **zero**, then we can say that there is **no relation** between two variables.

Covariance

- Recall that dot sums up the products of corresponding pairs of elements.
- When corresponding elements of x and y are either both above their means or both below their means, a positive number enters the sum.
- When one is above its mean and the other below, a negative number enters the sum.
- Accordingly, a "large" positive covariance means that x tends to be large when y is large and small when y is small.
- A "large" negative covariance means the opposite—that x tends to be small when y is large and vice versa.
- A covariance close to zero means that no such relationship exists.
- Nonetheless, this number can be hard to interpret, for a couple of reasons:
- Its units are the product of the inputs' units (e.g., friend-minutes-per-day), which can be hard to make sense of. (What's a "friend-minute-per-day"?)
- If each user had twice as many friends (but the same number of minutes), the covariance would be twice as large. But in a sense, the variables would be just as interrelated. Said differently, it's hard to say what counts a
- For this reason, it's more common to look at the correlation, which divides out the standard deviations of both variables.

```
# Sample data: Advertising budget and Sales revenue
```
Python program to compute covariance measures of any two variables.
```
advertising_budget = [10, 20, 30, 40, 50]  # Budget
sales_revenue_Pos = [200, 400, 600, 800, 1000]  # Revenue
sales_revenue_Neg = [2000,1500, 1200, 800, 100]  # Revenue
def calculate_covariance(x, y)->float:
    # Step 1: Calculate means of x and y
    mean_x = sum(x) / len(x)
    mean_y = sum(y) / len(y)
    # Step 2: Calculate covariance using the formula
    covariance = sum((x[i] - mean_x) * (y[i] - mean_y) for i in
range(len(x))) / len(x)
    return covariance
c=calculate_covariance(advertising_budget, sales_revenue_Pos)
if c== 0:
    print("Looks good!")
elif c < 0:
    print("Negative Covariance")
else:
    print("Positive Covariance")
# Print the result
print(f"Covariance between advertising budget and sales revenue:
{c:.2f}")
```

Correlation

- When two sets of data are strongly linked together we say they have a High Correlation.
- Correlation is Positive when the values increase together, and
- Correlation is Negative when one value decreases as the other increases
- A correlation is assumed to be linear (following a line).

Correlation

- Covariance is a measure to show the extent to which given two random variables change with respect to each other.
- Correlation is a measure used to describe how strongly the given two random variables are related to each other.

Correlation:

The **correlation coefficient** r is calculated by dividing the covariance by the product of the standard deviations of x and y:

$$r = \frac{\mathrm{Cov}(x, y)}{\sigma_x \sigma_y}$$

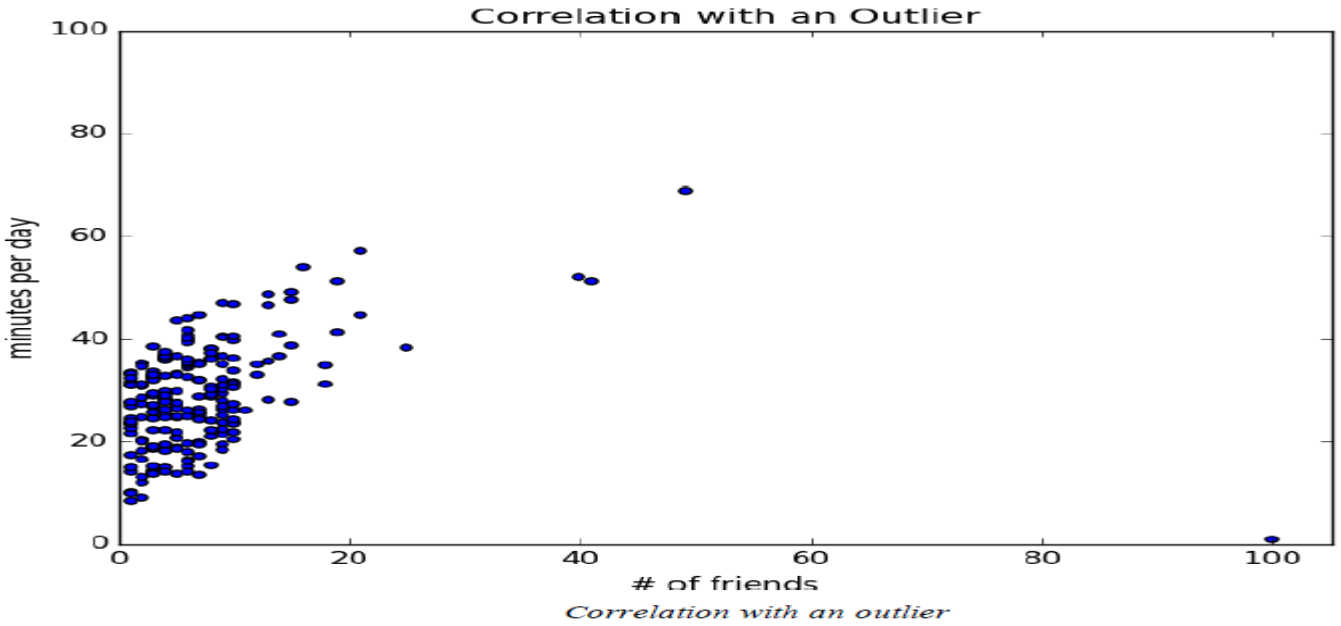
- r ranges between -1 and +1.
  - $r = 1$: Perfect positive correlation (they move together).
  - $r = -1$: Perfect negative correlation (they move oppositely).
  - $r = 0$: No linear relationship.

Correlation

```
import statistics
def calculate_correlation(x, y):
    if len(x) != len(y):
        raise ValueError("x and y must have the same length.")
    mean_x = statistics.mean(x)
    mean_y = statistics.mean(y)
    numerator = sum((x_i - mean_x) * (y_i - mean_y) for x_i, y_i in zip(x, y))
    denominator = ( (sum((x_i - mean_x) ** 2 for x_i in x) * sum((y_i - mean_y) ** 2 for y_i in y))  ** 0.5    )
    return numerator / denominator if denominator != 0 else 0
num_friends = [85,42,41,40,25,21,21,19,19,18,120,28,15,6,6,6,6,6,6,6,6,6]
daily_minutes =
[1,68.7,1.25,52.08,38.36,44.45,57.13,51.4,41.42,31.30,35,59,39.43,14.18,35.24,40.13,41.32,35.45,36.07,43.44,23.5
5,24.6]
correlation = calculate_correlation(num_friends, daily_minutes)
print(f"Pearson correlation coefficient: {correlation:.2f}")
if correlation <=0:
  print("No correlation!")
elif 0.1 < correlation < 0.25:
  print("Weak Correlation")
elif 0.25 < correlation < 0.5:
  print("Moderate Correlation")
elif 0.5 < correlation < 0.75:
  print("Strong Correlation")
elif 0.75 < correlation < 1:
  print("Very Strong Correlation")
else:
  print("Invalid")
```

Correlation

- The correlation is unitless and always lies between –1 (perfect anticorrelation) and 1 (perfect correlation).
- A number like 0.25 represents a relatively weak positive correlation.



Correlation with an outlier

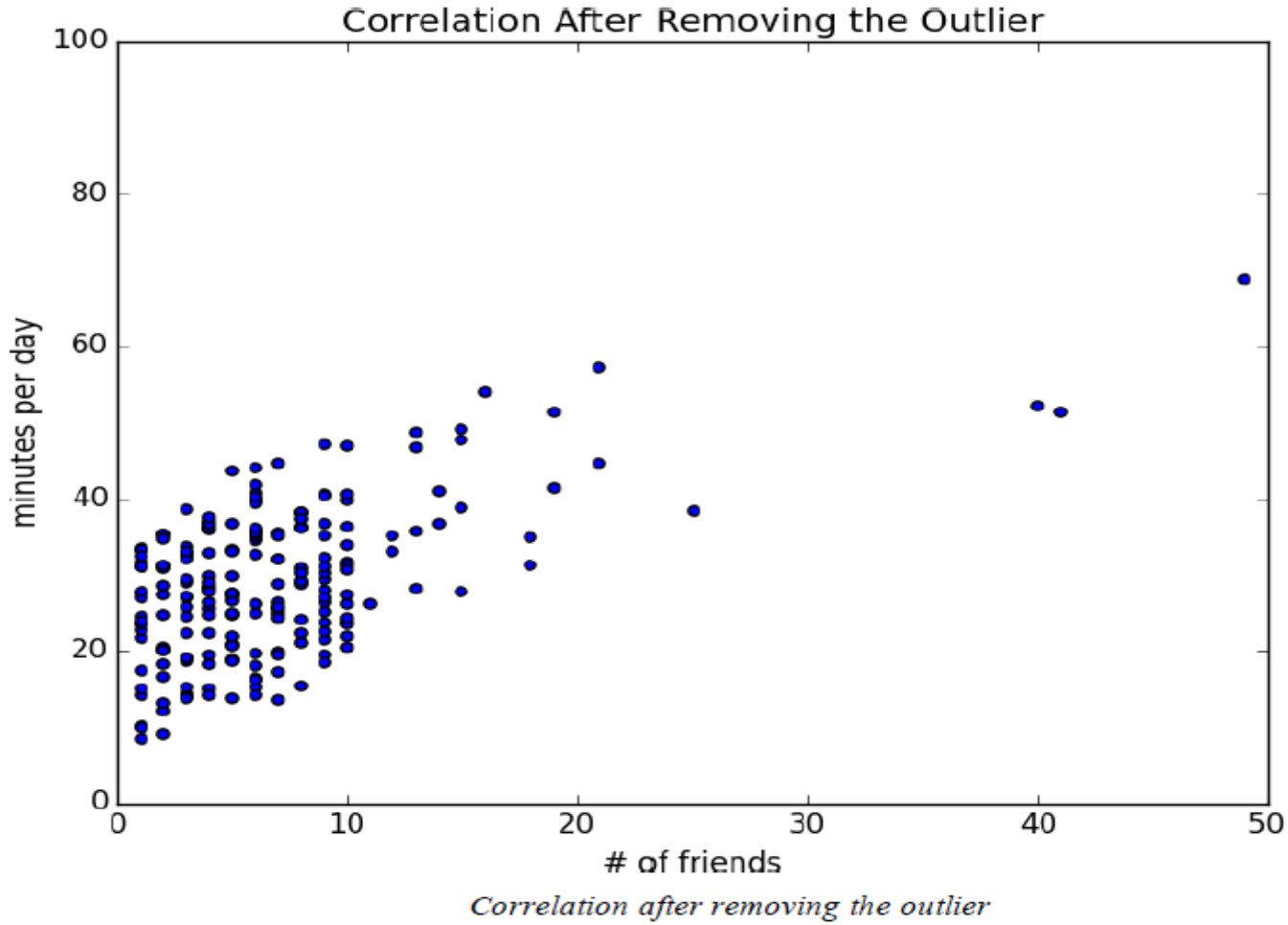
- Let's examine our data. The person with 100 friends (who spends only 1 minute per day on the site) is a huge outlier, and correlation can be very sensitive to outliers.
- What happens if we ignore him?

Correlation

```
import statistics
num_friends = [60,42,41,40,25,21,21,19,19,18,120,28,15,6,6,6,6,6,6,6,6,6]
outlier = num_friends.index(60) # index of outlier
print(outlier)
def calculate_correlation(x, y):
    """Calculates the Pearson correlation coefficient between two lists x and y."""
    if len(x) != len(y):
        raise ValueError("x and y must have the same length.")
    mean_x = statistics.mean(x)
    mean_y = statistics.mean(y)
    numerator = sum((x_i - mean_x) * (y_i - mean_y) for x_i, y_i in zip(x, y))
    denominator = ((sum((x_i - mean_x) ** 2 for x_i in x) * sum((y_i - mean_y) ** 2 for y_i in
y)) ** 0.5    )
    return numerator / denominator if denominator != 0 else 0
daily_minutes =
[1,68.7,1.25,52.08,38.36,44.45,57.13,51.4,41.42,31.30,35,59,39.43,14.18,35.24,40.13,41.32,35.4
5,36.07,43.44,23.55,24.6]
daily_minutes_good = [x for i, x in enumerate(daily_minutes) if i != outlier]
num_friends_good = [x for i, x in enumerate(daily_minutes) if i != outlier]
print(daily_minutes_good)
print(calculate_correlation(num_friends_good, daily_minutes_good))
```

Correlation without Outlier

- Without the outlier, there is a much stronger correlation.



Correlation after removing the outlier

Simpson's Paradox

- One common surprise when analyzing data is Simpson's paradox, in which correlations can be misleading when confounding variables (is an external variable that affects both the independent variable (the one you manipulate) and the dependent variable (the one you measure), creating a misleading or biased relationship between them.)are ignored.
- For example, imagine that you can identify all of your members as either East Coast data scientists or West Coast data scientists. You decide to examine which coast's data scientists are friendlier:

Coast	# of members	Avg. # of friends
West Coast	101	8.2
East Coast	103	6.5

- It certainly looks like the West Coast data scientists are friendlier than the East Coast data scientists.

Simpson's Paradox

- If you look only at people with PhDs, the East Coast data scientists have more friends on average. And if you look only at people without PhDs, the East Coast data scientists also have more friends on average!

Coast	Degree	# of members	Avg. # of friends
West Coast	PhD	35	3.1
East Coast	PhD	70	3.2
West Coast	No PhD	66	10.9
East Coast	No PhD	33	13.4

Simpson’s Paradox

- **Simpson's Paradox** occurs when a trend appears in different groups of data but disappears or reverses when the data is combined. This paradox demonstrates how misleading conclusions can arise if we fail to consider underlying subgroups in a dataset.
- In data science, Simpson’s Paradox is a critical reminder of the importance of analyzing data at multiple levels and understanding potential confounding factors that might affect the results.

Correlation and Causation

- **Causation** implies that one event directly affects another. It answers the question:
- **Does a change in one variable cause a change in another variable?**
- For causation to be established, you need to rule out confounding variables and establish a direct cause-and-effect relationship.
- So, ***"correlation is not causation,"***
- Nonetheless, this is an important point—if x and y are strongly correlated, that might mean that x causes y, that y causes x, that each causes the other, that some third factor causes both, or nothing at all.
- Consider the relationship between ***num_friends*** and ***daily_minutes***. It's possible that having more friends on the site causes DataSciencester users to spend more time on the site.
- **First might** be the case if each friend posts a certain amount of content each day, which means that the more friends you have, the more time it takes to stay current with their updates.
- **Second**, its possible that the more time users spend arguing in the DataSciencester forums, the more they encounter and befriend likeminded people.
- That is, spending more time on the site causes users to have more friends.
- A **third possibility** is that the users who are most passionate about data science spend more time on the site (because they find it more interesting) and more actively collect data science friends (because they don't want to associate with anyone else).

Probability

- Probability is a way of quantifying the uncertainty associated with events chosen from some universe of events.
- The universe consists of all possible outcomes.
- And any subset of these outcomes is an event; for example, “the die rolls a 1” or “the die rolls an even number.”
- Notationally, we write P(E) to mean “the probability of the event E.”
- We’ll use probability theory to build and evaluate models.

Dependence and Independence

- Two events E and F are dependent if knowing something about whether E happens gives us information about whether F happens (and vice versa).
- Otherwise, they are independent.
- For instance, if we flip a fair coin twice, knowing whether the first flip is heads gives us no information about whether the second flip is heads or tail. These events are independent.
- On the other hand, knowing whether the first flip is heads certainly gives us information about whether both flips are heads or not. These two events are dependent.
- Mathematically, we say that two events E and F are independent if the probability that they both happen is the product of the probabilities that each one happens:

$$P(E, F) = P(E)P(F)$$

Conditional Probability

Conditional probability measures the probability of an event occurring, given that another event has already occurred. It is a fundamental concept in probability theory and statistics.

The conditional probability of event A, given that event B has occurred, is denoted by $P(A \mid B)$ and is defined by the formula:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \quad \text{if} \quad P(B) > 0$$

Where:

- $P(A \mid B)$: Conditional probability of A given B
- $P(A \cap B)$: Joint probability that both A and B occur
- $P(B)$: Probability that event B occurs

Conditional Probability

Example:

Suppose you draw a card from a standard deck of 52 playing cards.

Let:

- Event A: The card drawn is a King
- Event B: The card drawn is a face card (King, Queen, or Jack)

In a standard deck:

- $P(A) = \frac{4}{52}$ (since there are 4 Kings)
- $P(B) = \frac{12}{52}$ (since there are 12 face cards: 4 Kings, 4 Queens, and 4 Jacks)
- $P(A \cap B) = \frac{4}{52}$ (since all 4 Kings are also face cards)

Now, calculate $P(A \mid B)$:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{4}{52}}{\frac{12}{52}} = \frac{4}{12} = \frac{1}{3}$$

So, given that the card drawn is a face card, the probability that it is a King is $\frac{1}{3}$.

Bayes Probability

The conditional probability of E given F, denoted $P(E|F)$, means the probability that event E happens, knowing that event F has already happened.

From the definition of conditional probability:

$$P(E|F) = \frac{P(E,F)}{P(F)}$$

Here:

- $P(E,F)$: The probability that both events E and F occur.
- $P(F)$: The probability that event F occurs.

Since $P(E,F)$ (joint probability of E and F) can also be written as:

$$P(E,F) = P(F|E) \cdot P(E)$$

This gives:

$$P(E|F) = \frac{P(F|E) \cdot P(E)}{P(F)}$$

This is the first key step toward Bayes' Theorem.

Bayes Probability

Now, we need to break down $P(F)$ (the probability that event F happens).

The event F can happen in two ways:

1. F happens *and* E happens (denoted $P(F, E)$).
2. F happens *and* E does *not* happen (denoted $P(F, \neg E)$).

So, we can write:

$$P(F) = P(F, E) + P(F, \neg E)$$

Using conditional probability, we can express these as:

$$P(F) = P(F|E) \cdot P(E) + P(F|\neg E) \cdot P(\neg E)$$

Here:

- $P(F|E)$: Probability that F happens given E.
- $P(F|\neg E)$: Probability that F happens given that E *does not* happen.
- $P(\neg E)$: Probability that E does not happen ($P(\neg E) = 1 - P(E)$).

Bayes Probability

Now, substitute $P(F)$ back into the conditional probability formula:

$$P(E|F) = \frac{P(F|E) \cdot P(E)}{P(F|E) \cdot P(E) + P(F|\neg E) \cdot P(\neg E)}$$

This is **Bayes' Theorem**, and it allows us to "reverse" conditional probabilities (i.e., switch E and F).

Bayes Probability

Simple Example: Medical Test for a Rare Disease

Imagine there is a rare disease that affects 1 out of every 10,000 people. You take a test that says you have the disease, but how accurate is that test really? Let's break it down using Bayes' Theorem.

- Event D: You have the disease.
- Event T: The test says you are positive for the disease.

We want to calculate: **What is the chance you actually have the disease if the test says you are positive?** This is $P(D|T)$, the probability of having the disease, given a positive test result.

Given Information:

1. $P(D)$: Probability that a random person has the disease = 1/10,000 = 0.0001.
2. $P(\neg D)$: Probability that a random person does *not* have the disease = 0.9999.
3. $P(T|D)$: Probability that the test is positive if you have the disease = 0.99.
4. $P(T|\neg D)$: Probability that the test is positive even if you don't have the disease (false positive) = 0.01.

Bayes Probability

Applying Bayes' Theorem:

Now, plug these values into the formula:

$$P(D|T) = \frac{P(T|D) \cdot P(D)}{P(T|D) \cdot P(D) + P(T|\neg D) \cdot P(\neg D)}$$

Substitute the numbers:

$$P(D|T) = \frac{(0.99) \cdot (0.0001)}{(0.99) \cdot (0.0001) + (0.01) \cdot (0.9999)}$$

$$P(D|T) = \frac{0.000099}{0.000099 + 0.009999}$$

$$P(D|T) = \frac{0.000099}{0.010098}$$

$$P(D|T) \approx 0.0098 \text{ or } 0.98\%$$

Conclusion:

Even though the test is 99% accurate, because the disease is so rare, the chance that you actually have the disease after testing positive is **only about 1%**. This is because false positives happen more frequently than true positives when dealing with rare conditions.

Random Variables

- A random variable is a variable whose possible values have an associated **probability distribution**. A very simple random variable equals 1 if a coin flip turns up heads and 0 if the flip turns up tails.
- A more complicated one might measure the number of heads you observe when **flipping a coin 10 times** or a value picked from range(10) where each number is equally likely.
- The associated distribution gives the probabilities that the variable realizes each of its possible values. The coin flip variable equals 0 with probability 0.5 and 1 with probability 0.5.
- The range(10) variable has a distribution that assigns probability 0.1 to each of the numbers from 0 to 9.

Random Variables

- This program simulates tossing two coins 10 times, each time recording the result of the tosses using random library. The goal is to keep track number of the following conditions: Both Coins Show Heads, First Coin Shows Heads, At Least One Coin Shows Heads.

```
import random
def coin_toss():
    return random.choice(["heads", "tails"])
both_heads = 0
at_least_one_head = 0
first_is_heads = 0
random.seed(0)
for _ in range(10):  # Simulate 10 tosses of two coins
    first_coin = coin_toss()
    second_coin = coin_toss()
    print(first_coin,second_coin)

    if first_coin == "heads":
        first_is_heads += 1
    if first_coin == "heads" and second_coin == "heads":
        both_heads += 1
    if first_coin == "heads" or second_coin == "heads":
        at_least_one_head += 1
print(f"Both are heads: {both_heads}")
print(f"First is heads: {first_is_heads}")
print(f"At least one is heads: {at_least_one_head}")
```

Continuous Distributions

- A coin flip corresponds to a **discrete distribution**—one that associates positive probability with discrete outcomes(whole numbers).
- **Continuous Distribution** is distributions across a continuous outcomes. (real numbers.)
- For example, the **uniform distribution** puts equal weight on all the numbers between 0 and 1.

```
def uniform_pdf(x):
  return 1 if x >= 0 and x < 1
else 0
print(uniform_pdf(-0.3))
```

Continuous Distributions

- There are infinitely many numbers between 0 and 1,.
- Represent a continuous distribution with a **probability density function (PDF)** such that the probability of seeing a value in a certain interval equals the integral of the density function over the interval.

1. Cumulative Distribution Function (CDF), gives the probability that a ***random variable is less than or equal to a certain value***.

```
def uniform_cdf(x):
  "returns the probability that a uniform random variable is <= x"
  if x < 0: return 0 # uniform random is never less than 0
  elif x < 1: return x # e.g. P(X <= 0.4) = 0.4
  else: return 1 # uniform random is always less than 1
print(uniform_cdf(0.5))
```

Continuous Distributions

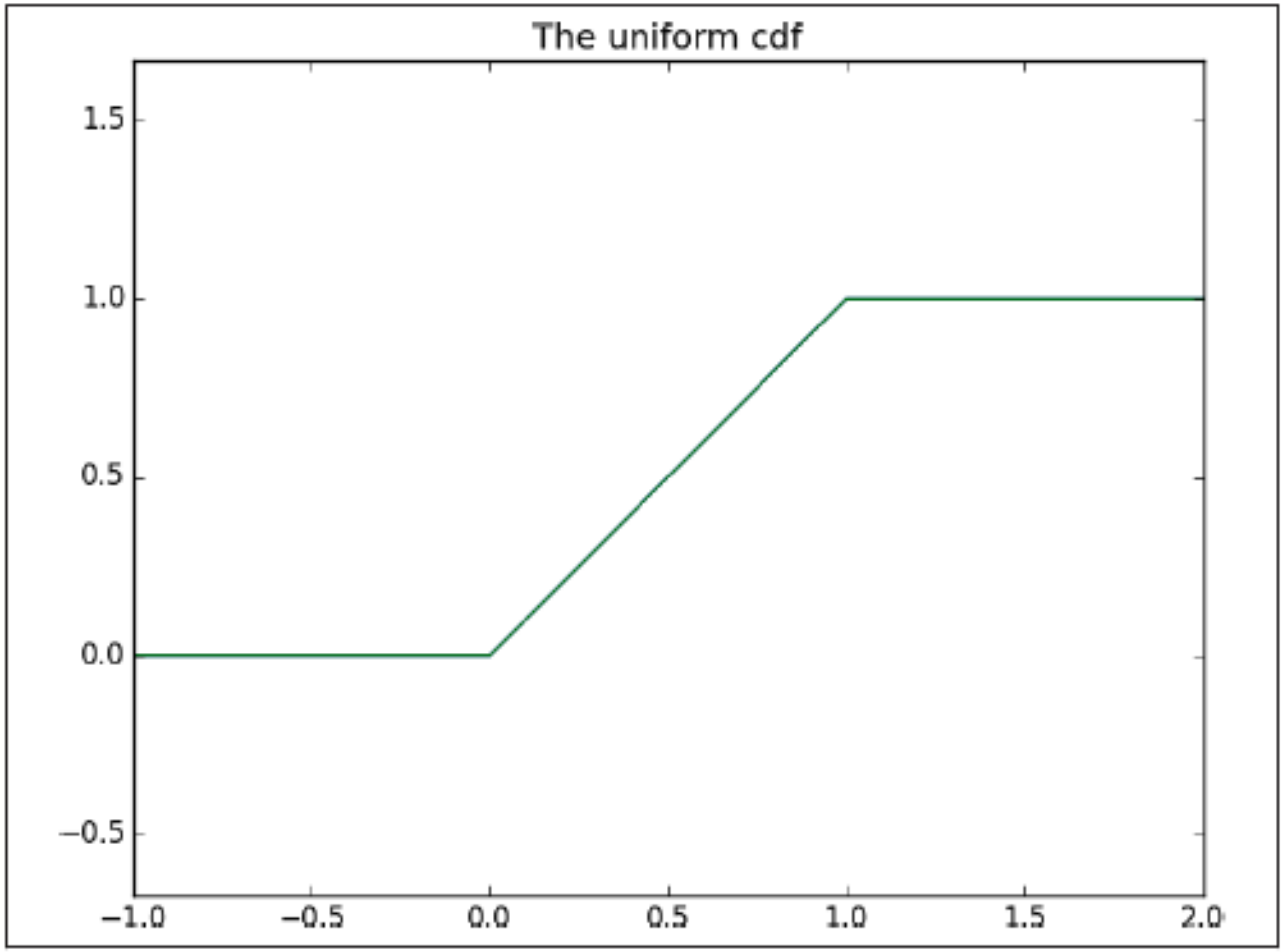


Figure 6-1. The uniform cdf

PDF for Normal Distribution

- The normal distribution (also called the Gaussian distribution) is a continuous probability distribution.
- The **PDF** describes the probability distribution of a continuous random variable at any specific value x in the interval.
- The normal distribution is the classic **bell curve–shaped** distribution and is completely determined by two parameters: its mean ***μ (mu)*** and its standard deviation ***σ (sigma).***
- The **mean** indicates where the bell is centered, and the **standard deviation** how "wide" it is.
- It has the PDF:

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The Normal Distribution

- It has the PDF:

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

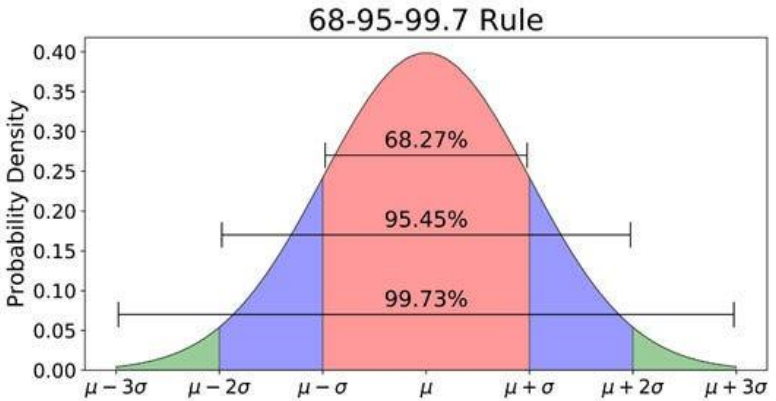
- Where:
- Probability density function value at
- Mean (average) of the distribution
- Standard deviation (a measure of spread or dispersion)

- The term $\frac{1}{\sigma\sqrt{2\pi}}$ is a **normalizing constant** that ensures the total area under the curve equals 1.
- The term $\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$ controls the shape of the curve:
  - If x is close to μ, the exponent is small, so the PDF value is high.
  - As x moves away from μ, the exponent becomes larger and the PDF value approaches zero.

The Normal Distribution

Key Characteristics of Normal Distribution:

- **Symmetry**: The distribution is symmetric around the mean .
- **Bell-Shaped Curve**: The highest point is at , and the curve tapers off as you move away from the mean.
  - A small σ makes the curve narrow and tall.
  - A large σ makes the curve wide and flat.
- **68-95-99.7 Rule (Empirical Rule):**
  - 68% of the data lies within 1 standard deviation ().
  - 95% of the data lies within 2 standard deviations ().
  - 99.7% of the data lies within 3 standard deviations ().Variance (square of the standard deviation)
- **PDF** describes the shape of a distribution and shows where values are most likely to occur.



The Normal Distribution

Need for Normal Distribution in Data Science:

1.Modeling Real-World Phenomena:

Many real-world processes follow normal distribution (e.g., height, IQ scores, measurement errors).

2.Anomaly Detection:

Deviations from the expected normal distribution can indicate anomalies (e.g., fraud detection).

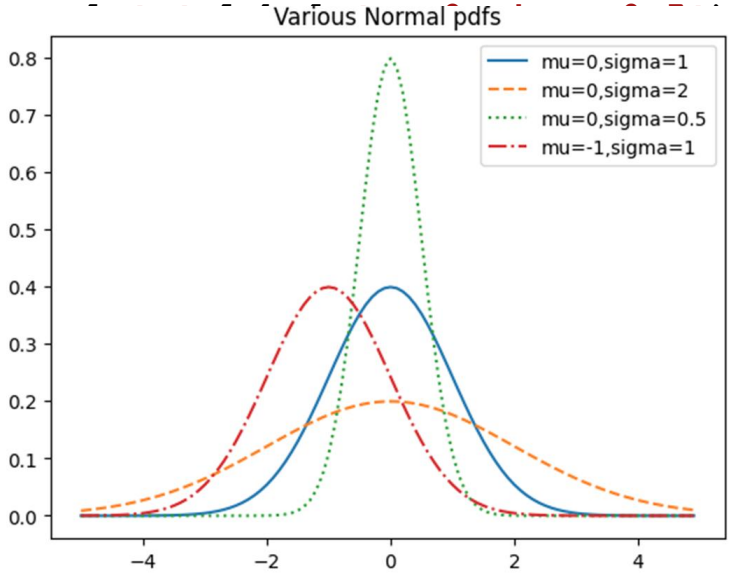
3.Error and Residual Analysis:

In regression models, normal distribution of residuals is a key assumption for accurate model performance.

The Normal Distribution

Python Program to demonstrate Various Normal PDF with varying values of mean and standard deviation

```
import math
def normal_pdf(x, mu=0, sigma=1):
  sqrt_two_pi = math.sqrt(2 * math.pi)
  return (math.exp(-(x-mu) ** 2 / 2 / sigma ** 2) / (sqrt_two_pi * sigma))
xs = [x / 10.0 for x in range(-50, 50)]
print(xs)
plt.plot(xs,[normal_pdf(x,sigma=1) for x in xs],'-',label='mu=0,sigma=1')
plt.plot(xs,[normal_pdf(x,sigma=2) for x in xs],'--',label='mu=0,sigma=2')
plt.plot(xs,[normal_pdf(x,sigma=0.5) for x i
plt.plot(xs,[normal_pdf(x,mu=-1) for x in xs
plt.legend()
plt.title("Various Normal pdfs")
plt.show()
```



The Normal Distribution

- When μ = 0 and σ = 1, it's called the **standard normal distribution**.
- If Z is a standard normal random variable, then it turns out that:

$$\mathbf{X = \sigma Z + \mu}$$

- is also normal but with mean μ and standard deviation σ. Conversely, if X is a normal random variable with mean μ and standard deviation σ,

$$\mathbf{Z = (X - \mu)/\sigma}$$

is a ***standard normal variable.***

- The CDF for the normal distribution cannot be written in an "elementary" manner, but we can write it using Python's ***math.erf*** error function:

CDF	$\frac{1}{2}\left[1+\operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$

The CDF for Normal Distribution

For a **normal distribution**, the CDF expression.

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)dt$$

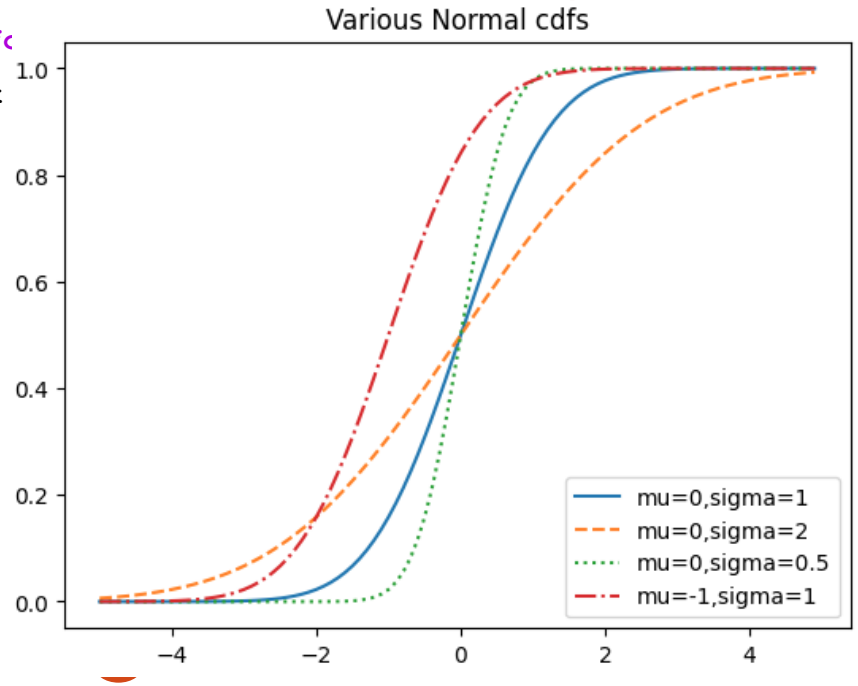
- The CDF for the normal distribution cannot be written in an "elementary" manner, but we can write it using Python's ***math.erf*** error function:

CDF	$\frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$

The Normal Distribution

Python Program to demonstrate CDF for Normal distribution using graph

```
from matplotlib import pyplot as plt
import math
def normal_cdf(x, mu=0,sigma=1):
  return (1 + math.erf((x - mu) / math.sqrt(2) / sigma)) / 2
xs = [x / 10.0 for x in range(-50, 50)]
plt.plot(xs,[normal_cdf(x,sigma=1) for x in xs],'-',label='mu=0,sigma=1')
plt.plot(xs,[normal_cdf(x,sigma=2) for x in xs],'--',label='mu=0,sigma=2')
plt.plot(xs,[normal_cdf(x,sigma=0.5) fo
plt.plot(xs,[normal_cdf(x,mu=-1) for x
plt.legend(loc=4) # bottom right
plt.title("Various Normal cdfs")
plt.show()
```



The Normal Distribution

•**PDF (Probability Density Function)**:
•Shows the relative likelihood of different delivery times.
•It does not give cumulative probabilities but instead shows how dense the distribution is at each point.
•**CDF (Cumulative Distribution Function)**:
•Used to calculate the cumulative probability that the delivery time is less than or equal to a certain value.
•Useful for finding probabilities over a range of values (like delivery between 30 and 40 minutes).

The Inverse Normal Distribution

- **Sometimes we'll need to invert *normal_cdf* to find the value corresponding to a specified probability.**
- **There's no simple way to compute its inverse, but *normal_cdf* is continuous and strictly increasing, so we can use a binary search**

```
Python Program to demonstrate CDF for Normal distribution using graph
import math

# Define the normal CDF
def normal_cdf(x, mu=0, sigma=1):
    """Returns the CDF of a normal distribution with mean mu and stddev sigma."""
    return (1 + math.erf((x - mu) / (sigma * math.sqrt(2)))) / 2
# Inverse of the normal CDF using binary search
```

The Inverse Normal Distribution

```
def inverse_normal_cdf(p, mu=0, sigma=1, tolerance=0.00001):
    # Check if p is within valid range
    if p <= 0 or p >= 1:
        raise ValueError("The probability p must be between 0 and 1.")
    # Define reasonable search bounds
    low_z, high_z = -10.0, 10.0
    # Use binary search to find inverse CDF
    while high_z - low_z > tolerance:
        mid_z = (low_z + high_z) / 2
        mid_p = normal_cdf(mid_z, mu, sigma)
        # Adjust search interval
        if mid_p < p:
            low_z = mid_z
        else:
            high_z = mid_z
    # Return the midpoint as the best estimate
    return (low_z + high_z) / 2
    # Find the value corresponding to the 97.5th percentile (p = 0.975)
z_value = inverse_normal_cdf(0.975)
print(f"The value corresponding to probability 0.975 is approximately {z_value:.5f}")

# For a normal distribution with mu = 50 and sigma = 10
z_value_custom = inverse_normal_cdf(0.975, mu=50, sigma=10)
print(f"The value corresponding to probability 0.975 with mu=50, sigma=10 is approximately
{z_value_custom:.5f}")
```

The Central Limit Theorem

- *Central Limit Theorem: A* random variable is defined as the average of a large number of independent and identically distributed random variables is itself approximately normally distributed.

In particular, if $x_1, ..., x_n$ are random variables with mean μ and standard deviation σ, and if n is large, then:

is approximately normally distributed with mean μ and standard deviation $\sigma/\sqrt{n}$. Equivalently (but often more usefully),

$$\frac{(x_1 + ... + x_n) - \mu n}{\sigma\sqrt{n}}$$

is approximately normally distributed with mean 0 and standard deviation 1.

The Central Limit Theorem

1. `bernoulli_trial(p)` – Performing a Single Bernoulli Trial

A **Bernoulli trial** is a random experiment with only two possible outcomes: **success (1)** or **failure (0)**.

- It's named after the Swiss mathematician Jacob Bernoulli.
- The trial has a fixed probability p of success.

  For example:

  - Flipping a coin: $p = 0.5$ (50% chance of getting heads).
  - Checking if a lightbulb is defective, where $p = 0.05$ is the probability it's defective.

2. `binomial(n, p)` – Counting the Number of Successes in n Trials

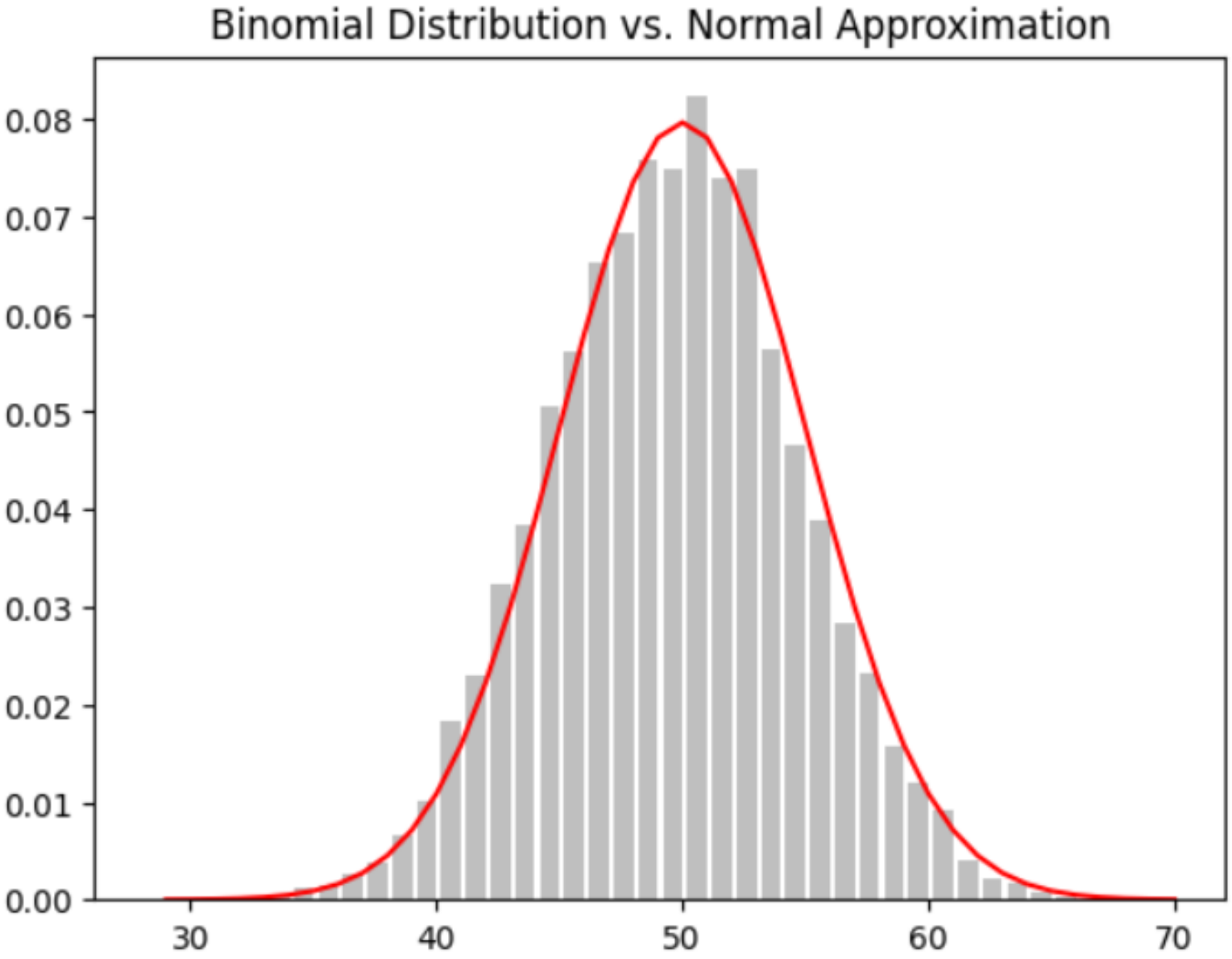
The **binomial distribution** models the number of successes in n independent Bernoulli trials, each with success probability p.

This function simulates n Bernoulli trials and returns the total number of successes.

The Central Limit Theorem

```
import random
import math
from collections import Counter
import matplotlib.pyplot as plt
# Function to perform a single Bernoulli trial with success probability p
def bernoulli_trial(p):
    return 1 if random.random() < p else 0
# Function to perform n Bernoulli trials and return the number of successes
def binomial(n, p):
    return sum(bernoulli_trial(p) for _ in range(n))
# Cumulative distribution function for a normal distribution
def normal_cdf(x, mu=0, sigma=1):
    return (1 + math.erf((x - mu) / math.sqrt(2) / sigma)) / 2
```

```
# Function to generate and plot the binomial and normal approximation
def make_hist(p, n, num_points):
    data = [binomial(n, p) for _ in range(num_points)]
    # Create a histogram of the binomial data
    histogram = Counter(data)
    plt.bar( [x - 0.4 for x in histogram.keys()], [v / num_points for v in
histogram.values()], 0.8,
        color='0.75‘  )
        # Calculate mean and standard deviation for the normal approximation
    mu = p * n
    sigma = math.sqrt(n * p * (1 - p))
    # Plot the normal approximation as a line chart
    xs = range(min(data), max(data) + 1)
    ys = [normal_cdf(i + 0.5, mu, sigma) - normal_cdf(i - 0.5, mu, sigma) for i in xs]
    plt.plot(xs, ys, color='red')
    plt.title("Binomial Distribution vs. Normal Approximation")
    plt.show()
# Example usage
make_hist(p=0.5, n=100, num_points=10000)
```


Binomial Distribution vs. Normal Approximation
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
30
40
50
60
70