Python for Biologist Overview

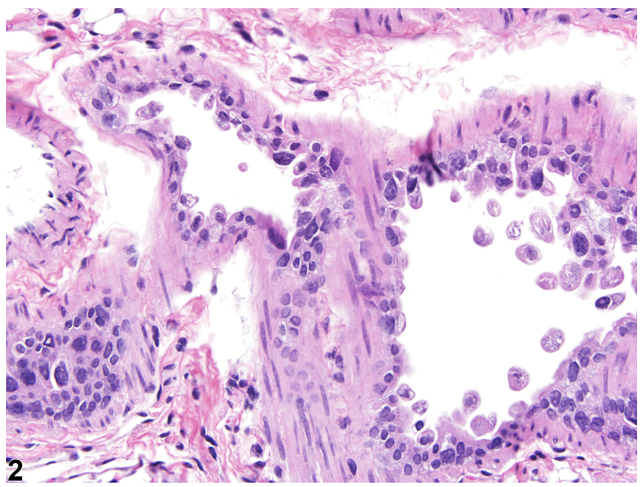
What is Data?

Data is information collected to observe, measure, and analyze phenomena.

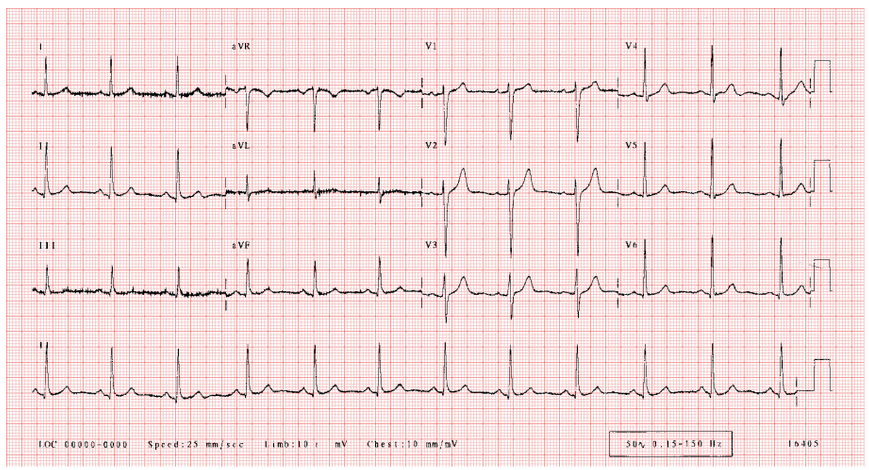
Types of Data

Quantitative Data: Numerical measurements (e.g., heart rate, enzyme activity, gene expression levels).

Qualitative Data: Descriptive data that categorizes observations (e.g., cell types, tissue classification, species).



Mouse Lung Epithelium



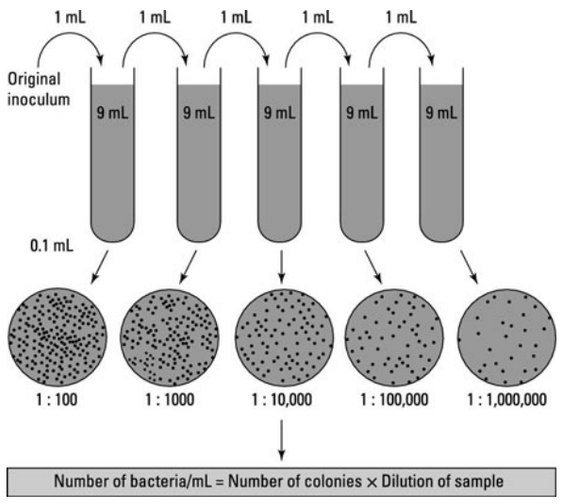
Human ECG

Why Collect Data?

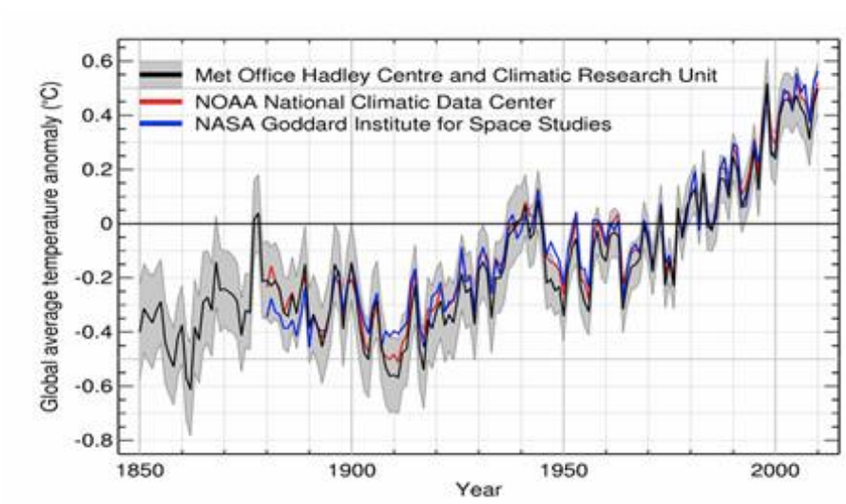
To observe patterns (e.g., population trends, genetic variations).
To test hypotheses (e.g., drug efficacy, behavioral responses).
To create models (e.g., ecological models, genetic models).

Examples in Biology:

Cell Counts



Global Surface Temperatures



Bee Waggle Dance

Data’s Journey

Collection → Analysis → Interpretation

Key Concepts to Remember

Data can be **structured** (e.g., tables with rows and columns) or **unstructured** (e.g., images, free-text observations).

Data Types in Python (or any coding language)

Categorical Data: Data with discrete groups or labels (e.g., species type, sex, tissue type). Boolean, nominal, or ordinal.

In Python: import pandas as pd
# Create a sample dataframe
data = {'color': ['red', 'blue', 'green', 'red', 'blue']}
df = pd.DataFrame(data)
# Convert 'color' column to categorical variable
df['color'] = df['color'].astype('category')

Numerical Data: Data expressed in numbers (e.g., weight, temperature). Integer, double, float

In Python: age = 25 # integer
height = 1.75 # float
heights = [1.75, 1.80, 1.81, 1.70] #multiple samples in a vector

Cleaning Data

Real-world data, we often encounter missing or inconsistent data that needs cleaning.

Missing Data, Standardizing Formats, Correct Errors, Removing Outliers, Validating Data, etc.

Python has libraries to aid this process:

Python: import pandas as pd
df = pd.DataFrame(…) # dataset with missing data (“na”)
 # Fill missing values with a scalar
df_filled = df.fillna(0) #use df.dropna() to drop missing data

Numerical Data: Continuous vs. Discrete

Continuous Data

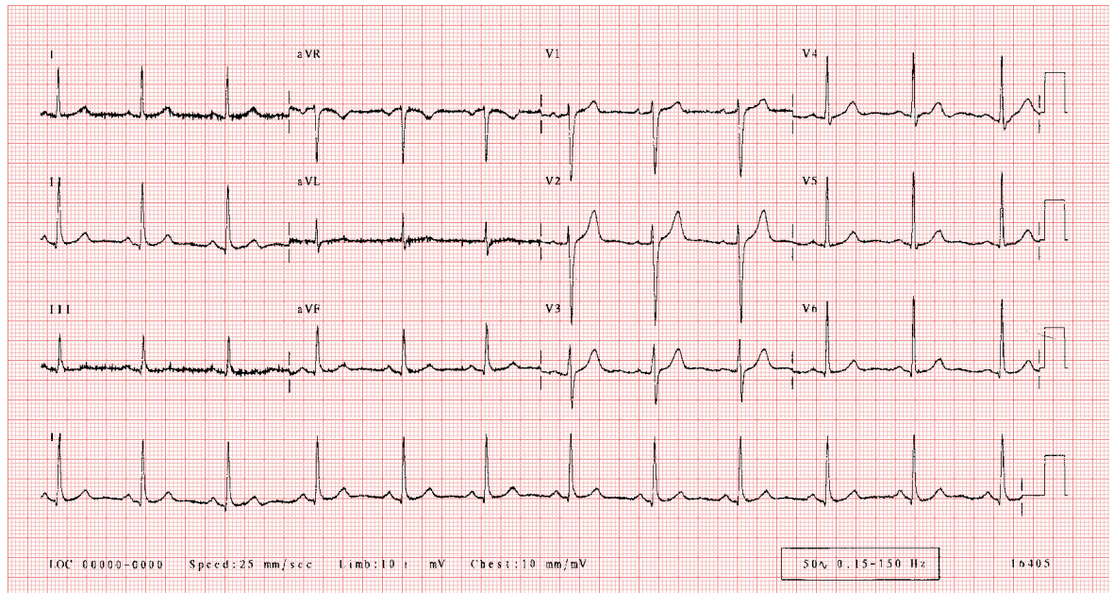
Definition: Data that can take any value within a range (e.g., height, temperature).

Example: Temperature in a biological reaction that could be measured at many precise points.

Discrete Data

Definition: Data with specific set values, often counted data (e.g., number of leaves on a plant).

Continuous data (like time) becomes discretized when stored in arrays.



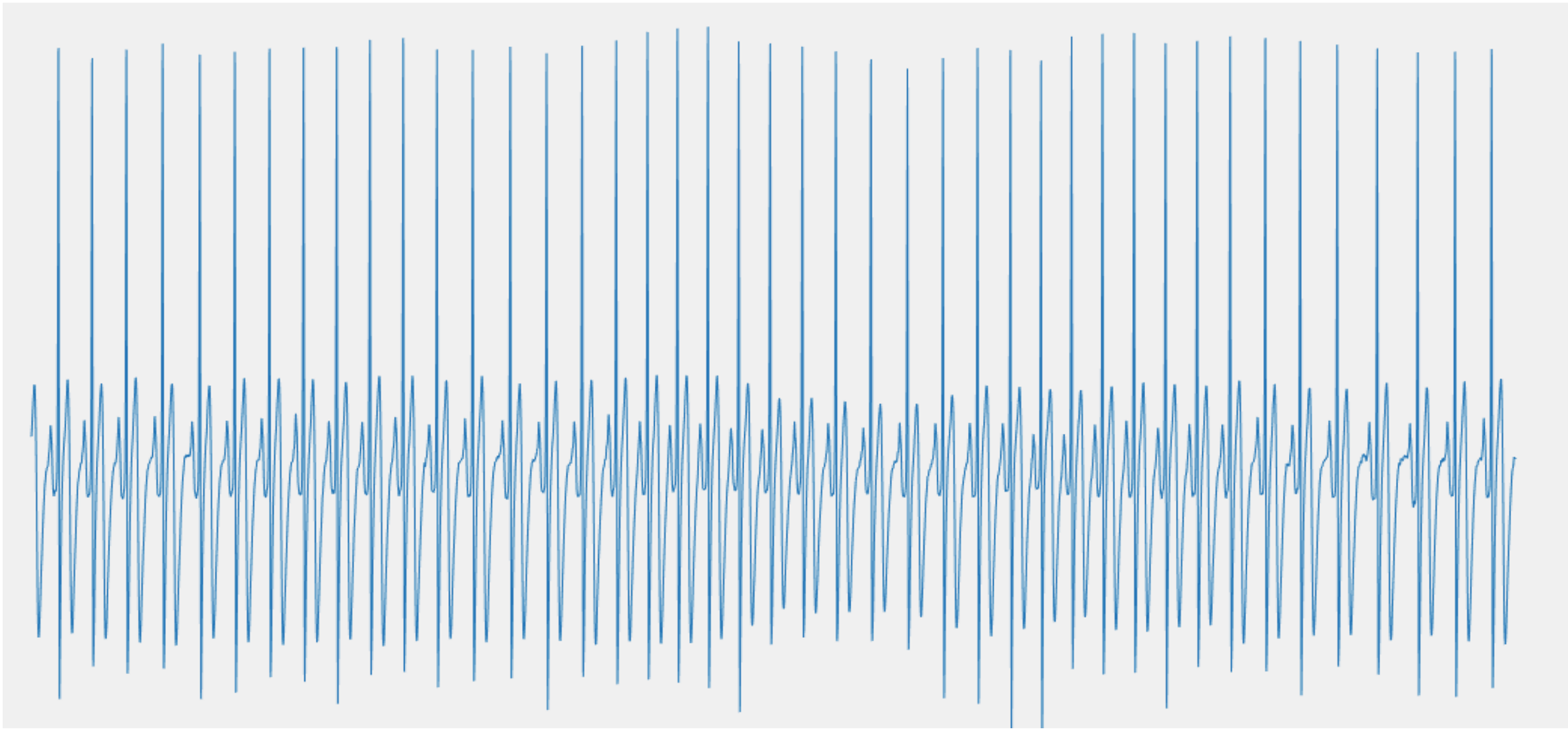
Human ECG

Sampling and Sample Rate

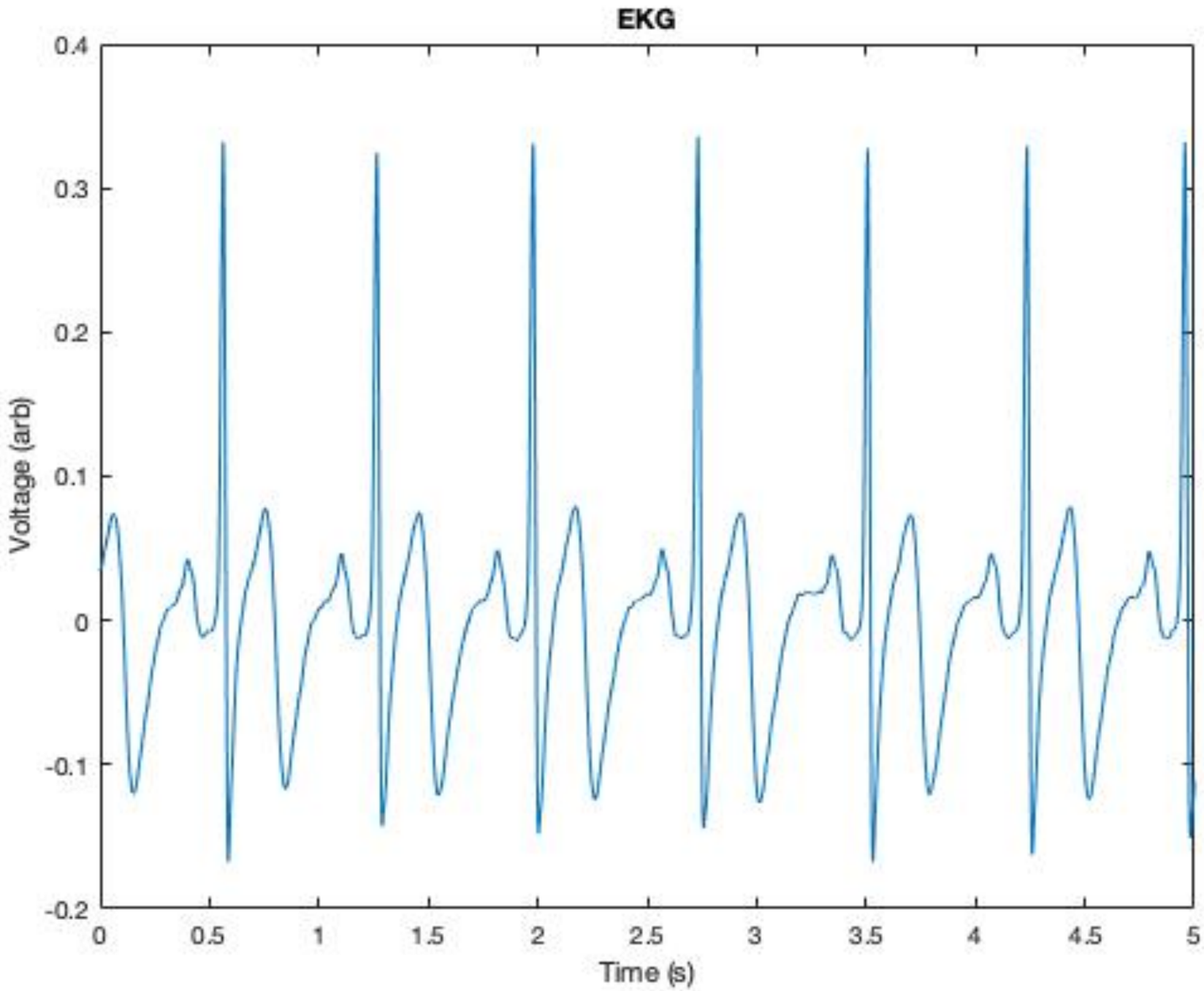
Sampling: In order to acquire continuous data, like the ECG, EMG, and EEG, we must collect data points at set intervals or specific moments.

Sample Rate is the interval at which we use to take our measurements. The sample rate allows for time calculation in a dataset.

Sampling Example



Sampling Example



Sampling Example

Python:

```
import numpy as np
[data, SR] = … #import data
# Calculate timestamps
timestamps = np.arange(len(data)) / SR
```

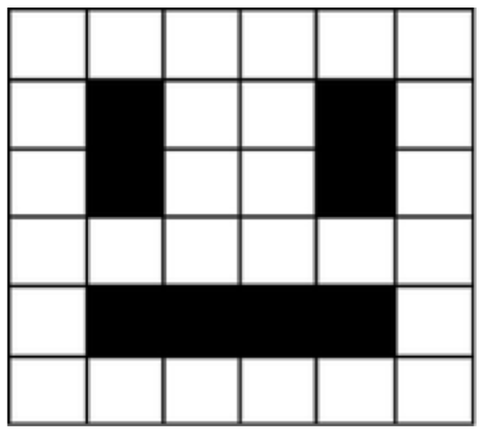
Explanation:
np.arange(len(data)) creates an array of indices for each point in data (e.g., [0, 1, 2, 3, 4]).

Dividing each index by SR converts these indices into timestamps in seconds, based on the sample rate.

Note: The timestamps and data arrays must always be the same length. If you remove a value from the data during cleaning, be sure to remove the corresponding timestamp at the same index to keep them aligned.

Sampling Example: Images and Videos

Images are represented as 2D arrays (or matrices), where each value corresponds to the intensity of a pixel. For an 8-bit grayscale image, intensity values range from 0 to 255, with 0 as black, 255 as white, and values in between representing shades of gray. Color images are represented as 3D matrices, with separate 2D layers (often for red, green, and blue channels) that combine to create a full-color image.



“binary pixel art” where 0 represents white

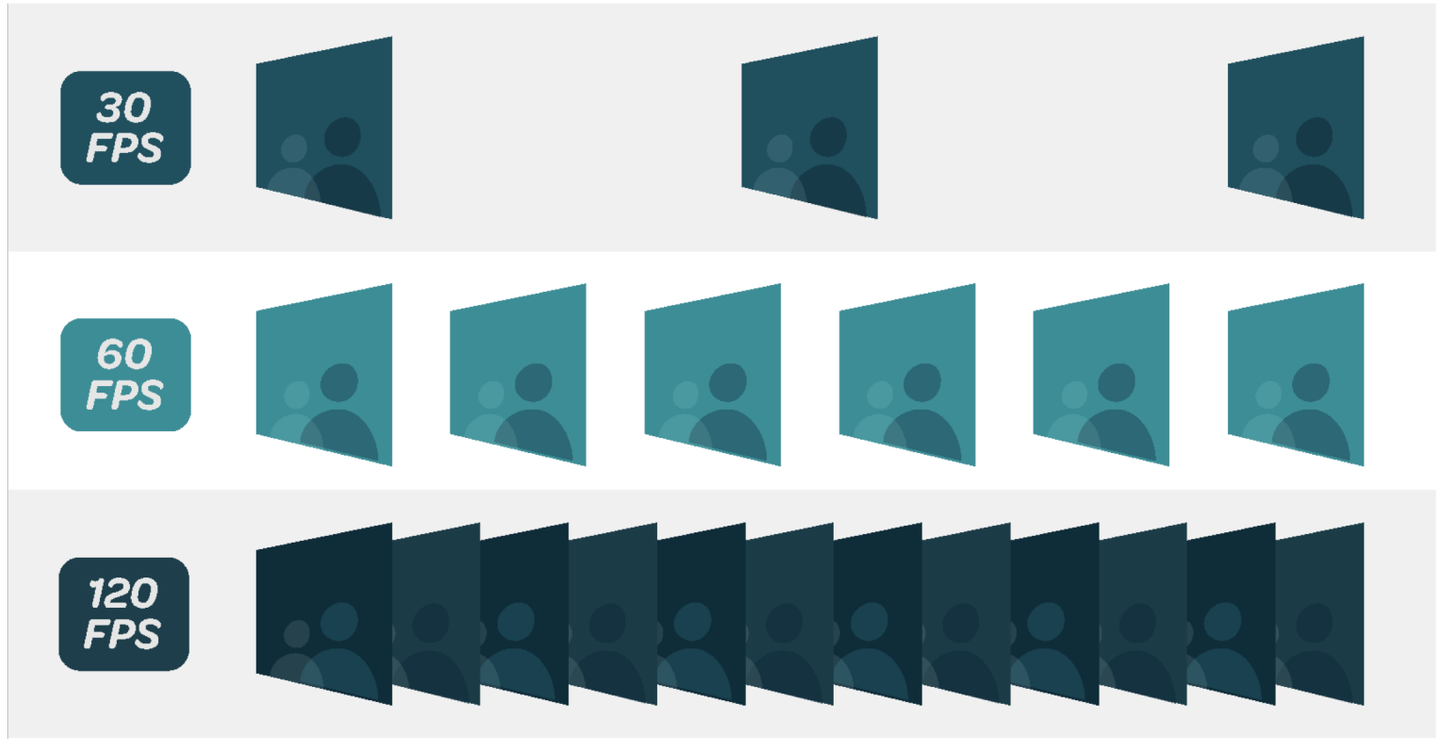
0	0	0	0	0	0
0	1	0	0	1	0
0	1	0	0	1	0
0	0	0	0	0	0
0	1	1	1	1	0
0	0	0	0	0	0

000000
010010
010010
000000
011110
000000

Matrix representation

Sampling Example: Images and Videos

Videos consist of images captured at a specific frame rate, meaning the number of frames (images) per second. A grayscale (black and white) video can be represented as a 3D matrix with dimensions for width, height, and frame number (x, y, frame). A color video, meanwhile, is represented as a 4D matrix with dimensions for width, height, frame number, and color channel (x, y, frame, color channel).



An example of different frame rates (frames per second)

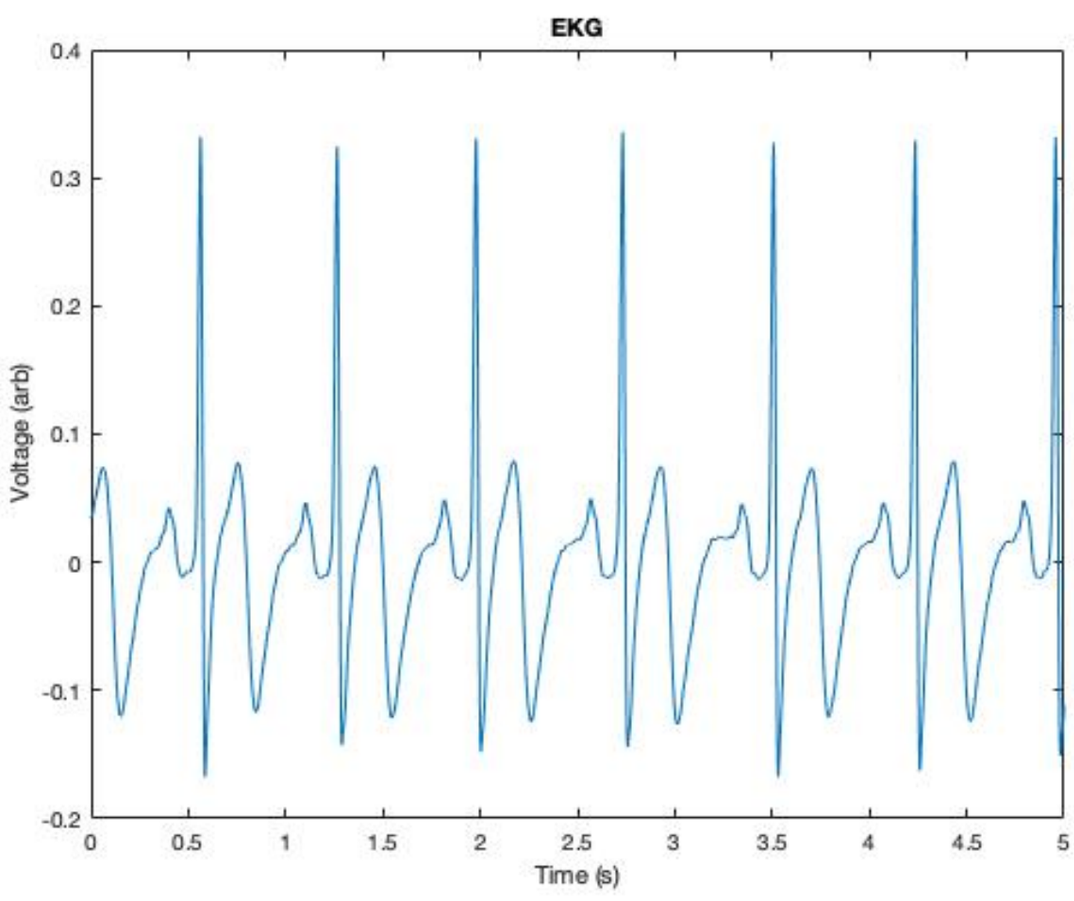
Data Visualization Basics

Visualization helps identify trends, outliers, and data distributions, which are important for interpreting biological results.

- Histograms for data distribution, **scatter plots** for examining relationships. In Python: use “matplotlib” or “seaborn.”

Note: In this graph I have labeled my axis appropriately, converted the x axis in time and given the figure a meaningful title.

```
import matplotlib.pyplot as plt
# Create a plot
plt.plot(timestamps, data)

#Set the current x-axis limits
plt.xlim(#lower lim, #upper lim)

# Add title and axis labels
plt.title("Sample Plot Title")
plt.xlabel("X-Axis Label")
plt.ylabel("Y-Axis Label")

plt.show()
```



Transforming Data

AI and Machine Learning are becoming mainstream in in biological data analysis. It is commonly used for:

image analysis,
predictive modeling in epidemiology,
genomics and proteomics,
protein structure prediction,
natural language processing,
drug discovery, signal processing,
ecology,
behavioral analysis …

Common Machine Learning Techniques

- **Supervised Learning**: Learning from labeled data (e.g., classifying types of cells).
  - Linear regression, Logistic Regression, Decision Trees, Random Forests, Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), Neural Networks, Gradient Boosting Machines (GBM), Naive Bayes

Unsupervised Learning: Finding patterns in unlabeled data (e.g., clustering similar gene expression profiles).

- k-Means Clustering, Hierarchical Clustering, Principal Component Analysis (PCA), Independent Component Analysis (ICA), t-Distributed Stochastic Neighbor Embedding (t-SNE), Autoencoders, Gaussian Mixture Models (GMM), Association Rule Learning (e.g., Apriori Algorithm)

Preprocessing Data

Signal Processing and Feature Extraction
Noise Reduction: Filtering is commonly used to remove noise from signals, such as ECG, EEG, or imaging data, to obtain cleaner data for analysis.
Enhancing Features: Filters can highlight specific features in the data (e.g., edges in images or frequency bands in EEG signals), making it easier for downstream algorithms to detect patterns.

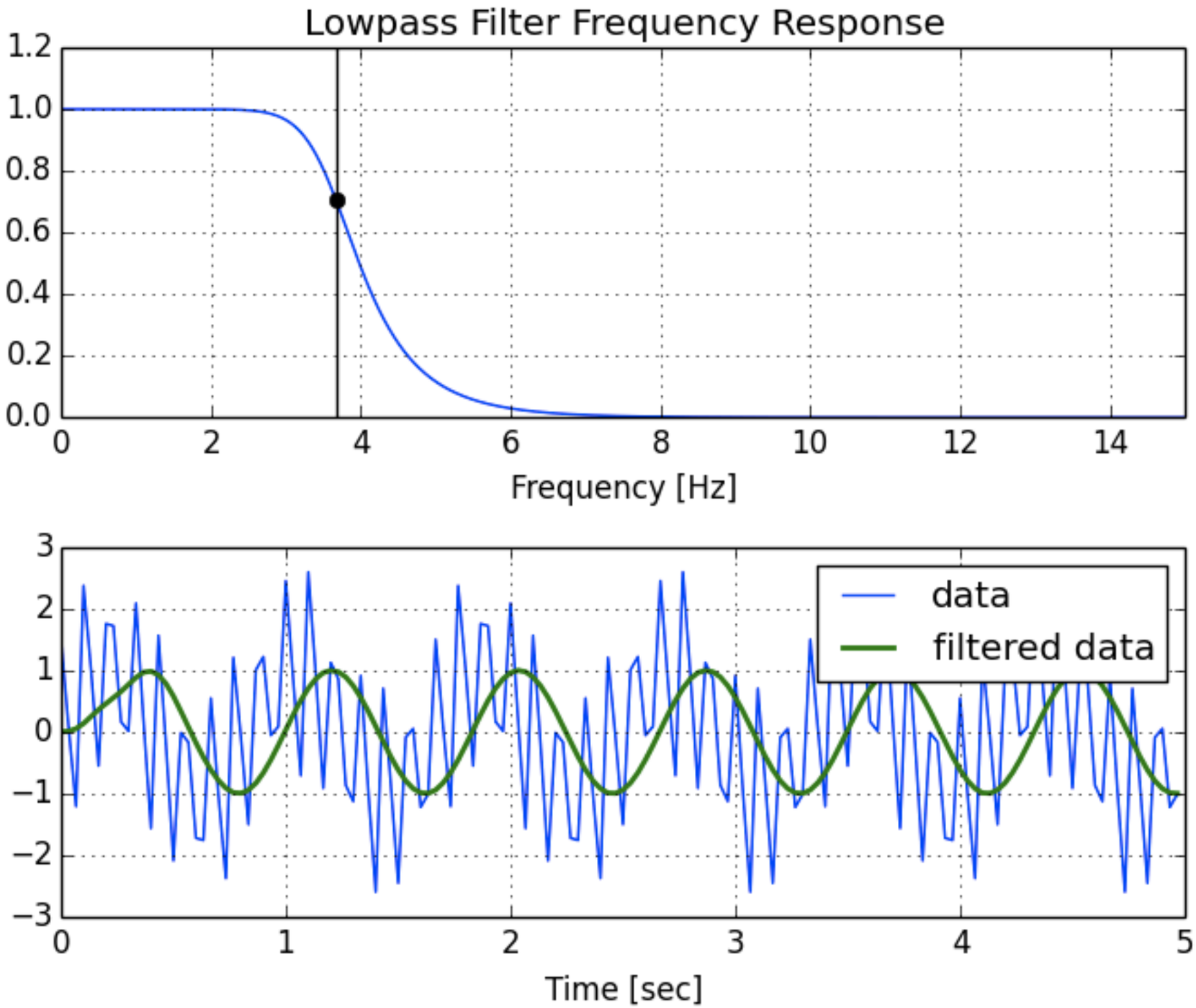
Types of Filtering in Data Analysis
Low-Pass Filters: Allow signals below a certain frequency to pass through, commonly used to remove high-frequency noise.
High-Pass Filters: Allow high-frequency signals through and remove low-frequency components, often used to filter out slow trends.
Band-Pass Filters: Allow a specific frequency range to pass, helpful in isolating specific signal components (e.g., brainwave frequencies).
Moving Average Filters: Smooth out short-term fluctuations, useful for trend analysis in time-series data.

Preprocessing Data

Filtering in Machine Learning Pipelines
Preprocessing Step: In a machine learning workflow, filtering can be part of data preprocessing. For example, before training a model, you might filter out noise in time-series data or smooth out spikes in sensor data.
Feature Engineering: Filtered data can be transformed into features, such as extracting certain frequency bands from a signal to represent distinct biological or behavioral characteristics.
Dimensionality Reduction: Filtering techniques like wavelet decomposition or Fourier transforms can help reduce data dimensionality by isolating important signal components, which are then fed into models.

Examples of Filtering in Biology and Data Science
Image Processing: Filtering is used to enhance features (e.g., edge detection in microscopy images), which can improve the performance of image classification models.
Time-Series Analysis: In biological signals like ECG or EEG, filtering is applied to focus on relevant frequency bands or remove artifacts before applying clustering or classification techniques.
Data Smoothing: Smoothing is a simple form of filtering, often used to reduce noise in datasets before applying machine learning algorithms.

Lowpass Filter Example



ML Example: Support Vector Machine (SVM)

Collection → **Analysis** → **Interpretation**

Collection → Dataset - image, 1D array, spreadsheets

Analysis → Choosing a method – preprocess – apply ML

Interpretation → statistics

Example:

What is the difference between a muffin and a cupcake? How can we find the answer?

ML Example: SVM

Example muffin recipe:

```
Basic Muffin Recipe
Ingredients

2 cups (260 g) all-purpose flour
½ cup (100 g) granulated sugar
2 teaspoons baking powder
½ teaspoon salt
¾ cup (180 ml) milk, room temperature
½ cup (114 g) unsalted butter, melted and cooled
2 large eggs, room temperature
2 tablespoons coarse sugar, optional
```

ML Example: SVM

Normalize:

Each row adds to 100

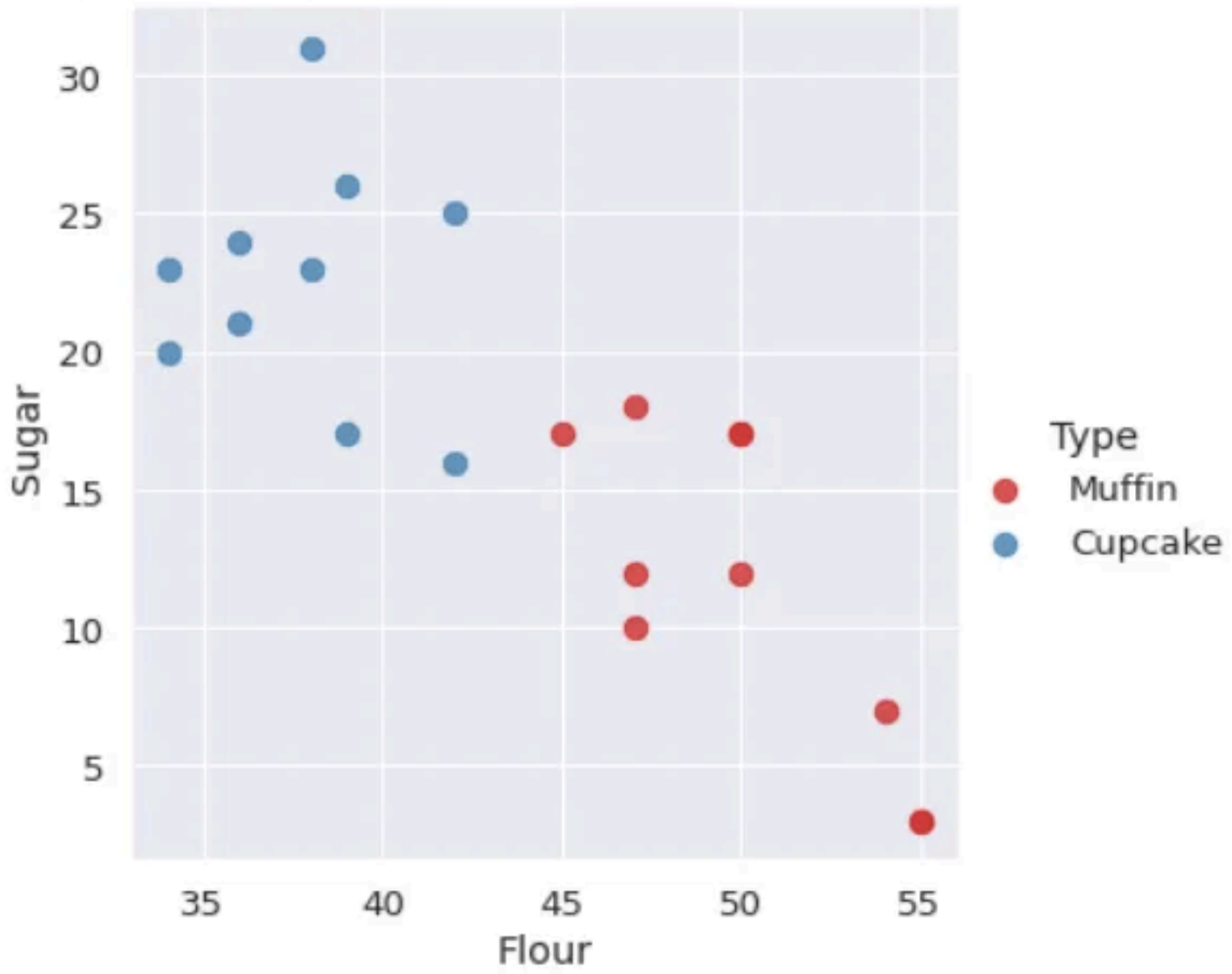
```
df = pd.read_csv('/recipes_muffins_cupcakes.csv')
df
```

	Type	Flour	Milk	Sugar	Butter	Egg	Baking Powder	Vanilla	Salt
0	Muffin	55	28	3	7	5	2	0	0
1	Muffin	47	24	12	6	9	1	0	0
2	Muffin	47	23	18	6	4	1	0	0
3	Muffin	45	11	17	17	8	1	0	0
4	Muffin	50	25	12	6	5	2	1	0
5	Muffin	55	27	3	7	5	2	1	0
6	Muffin	54	27	7	5	5	2	0	0
7	Muffin	47	26	10	10	4	1	0	0
8	Muffin	50	17	17	8	6	1	0	0
9	Muffin	50	17	17	11	4	1	0	0
10	Cupcake	39	0	26	19	14	1	1	0
11	Cupcake	42	21	16	10	8	3	0	0
12	Cupcake	34	17	20	20	5	2	1	0
13	Cupcake	39	13	17	19	10	1	1	0
14	Cupcake	38	15	23	15	8	0	1	0
15	Cupcake	42	18	25	9	5	1	0	0
16	Cupcake	36	14	21	14	11	2	1	0
17	Cupcake	38	15	31	8	6	1	1	0
18	Cupcake	36	16	24	12	9	1	1	0
19	Cupcake	34	17	23	11	13	0	1	0

ML Example: SVM

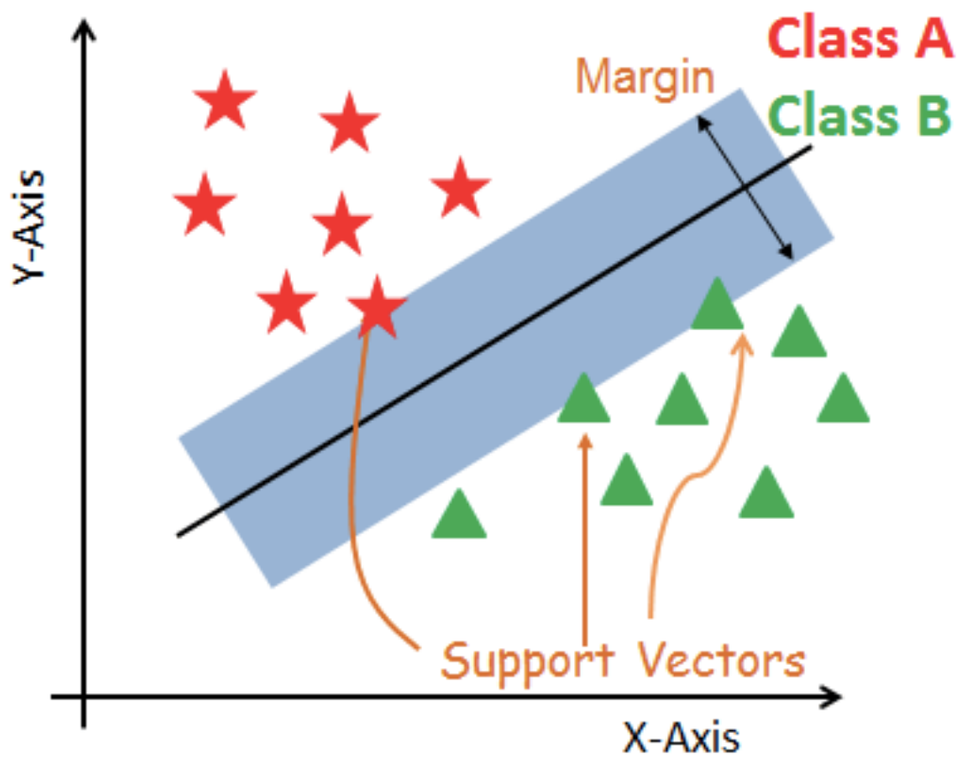
```
# Plot two ingredients
sns.lmplot('Flour', 'Sugar', data=df, hue='Type',
           palette='Set1', fit_reg=False, scatter_kws={"s": 70})
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43
  FutureWarning
```



ML Example: SVM

- **Support Vectors**
- Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.
- **Hyperplane**
- A hyperplane is a decision plane which separates between a set of objects having different class memberships.
- **Margin**
- A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.



ML Example: SVM

Build a training data set by holding some data back. *different dataset with simpler code to understand*

Python:
Import train_test_split function from sklearn.model_selection import train_test_split
Split dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.3,random_state=109)
70% training and 30% test

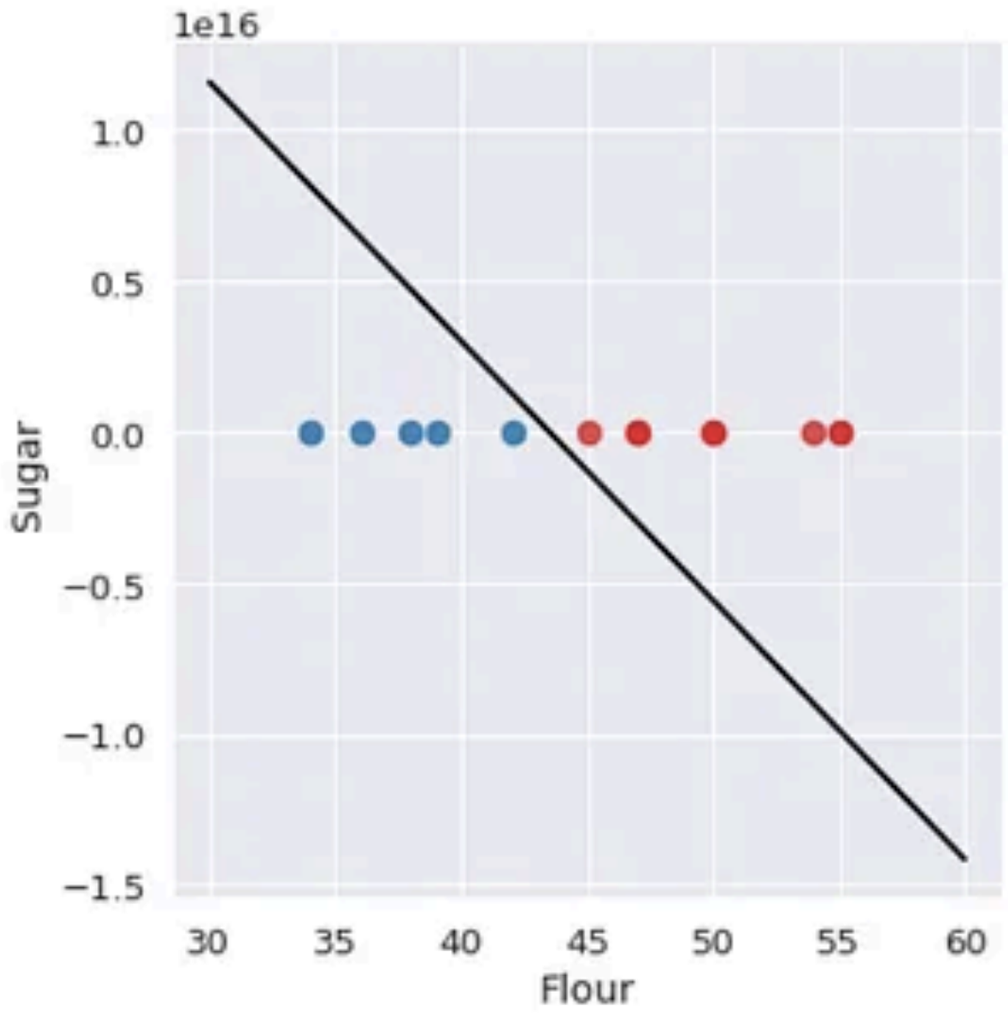
https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

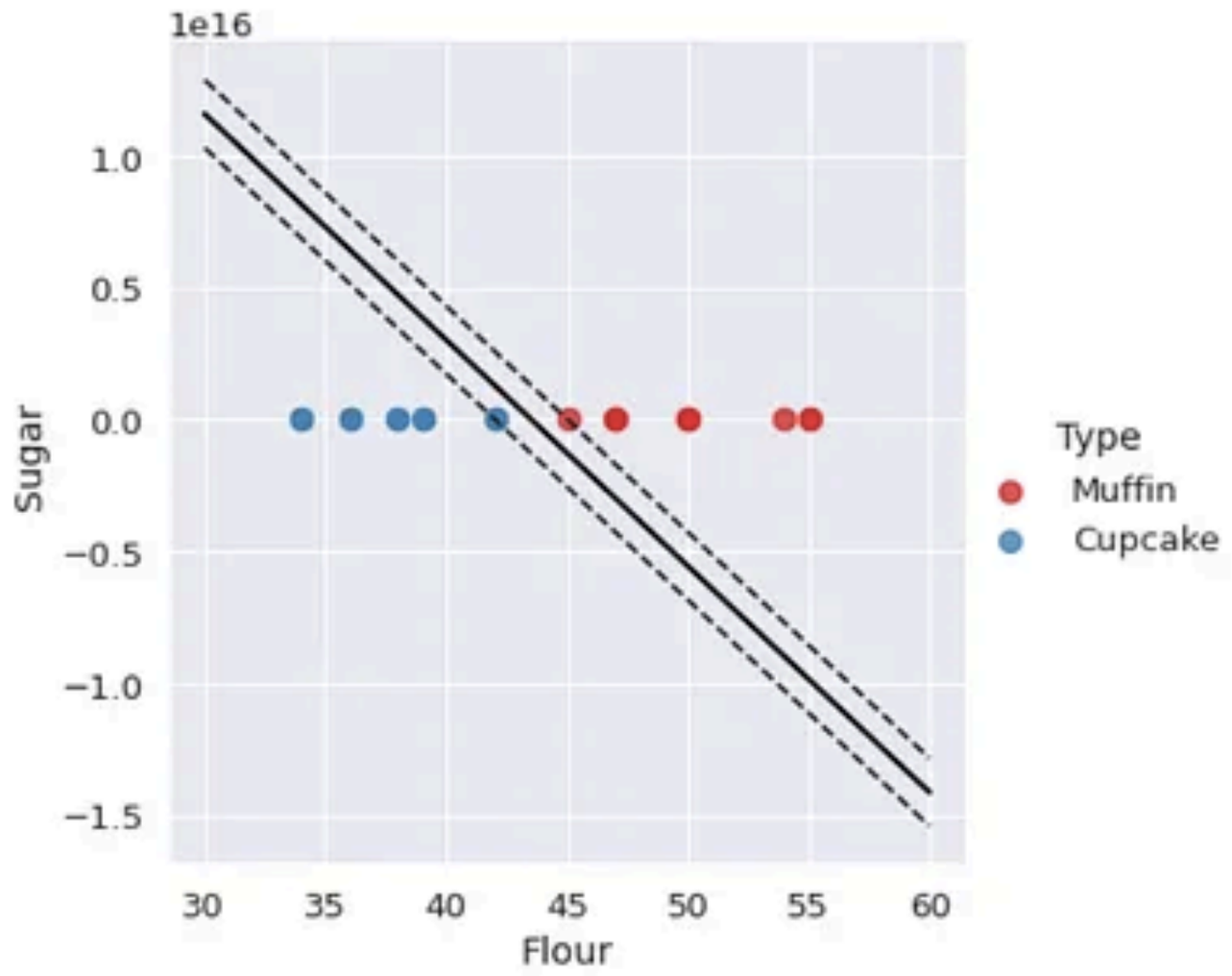
SVM Code!!

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear’) # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

ML Example: SVM

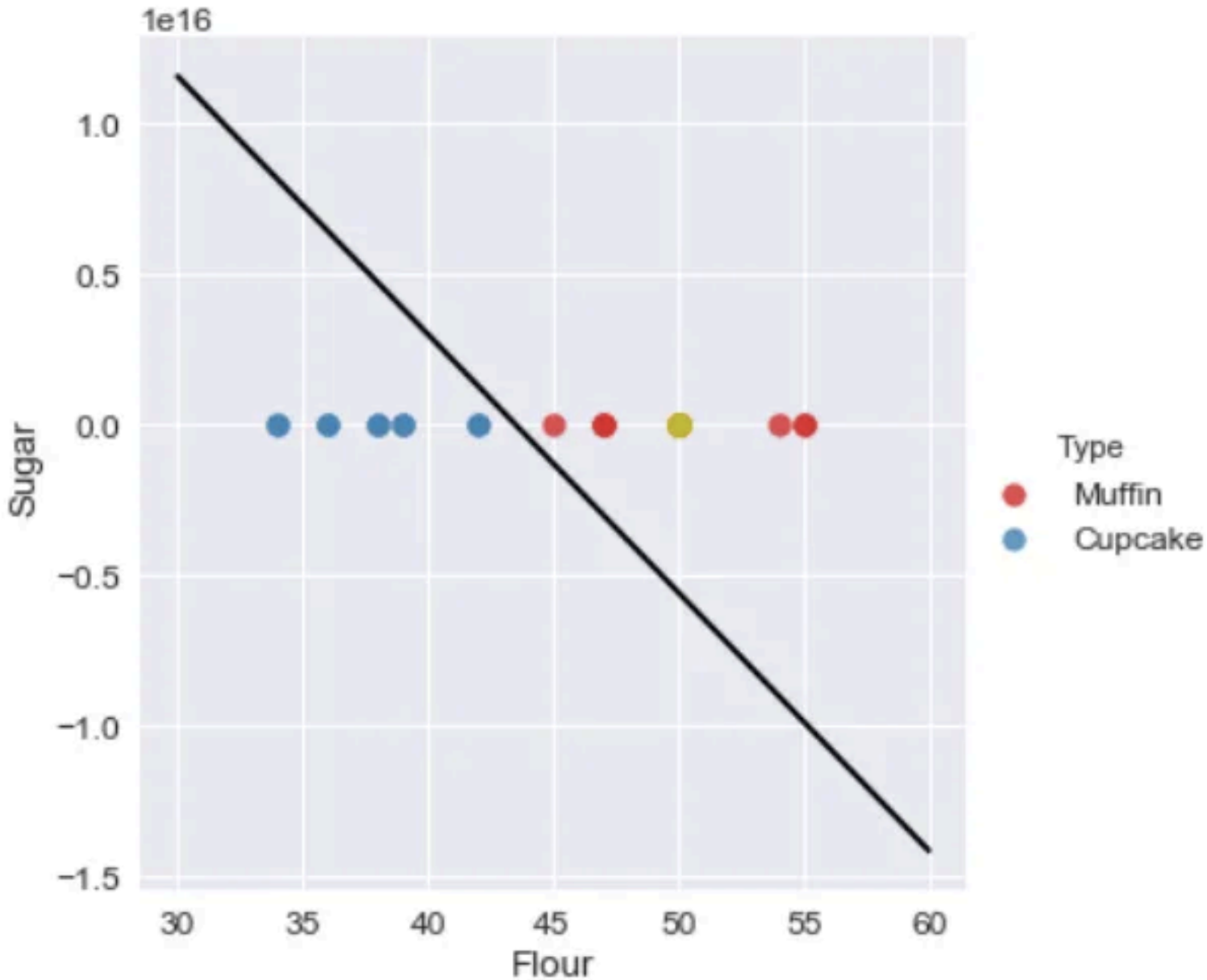
Results:





ML Example: SVM

Predict if 50 parts flour and 20 parts sugar



Statistics in Data Analysis

- **Purpose of Statistics**
  - Statistics are used in data analysis, identifying trends, significance testing, and making predictions.
    - Hypothesis testing, confidence intervals, correlation, etc.
- **Python Simplifies Statistical Analysis**
  - Packages: pandas, numpy, and scipy
  - Make simple graphs, bar plots, and simple statistical analysis: t-test, anova, etc.

Steps in the Data Analysis Pipeline

Collection → Analysis → Interpretation

Import Data

Begin by loading your dataset into the analysis environment (e.g., CSV files, Excel files, or direct database connections).

Get the Timestamps

Calculate or extract timestamps for each data point, especially critical in time-series or experimental data.

Plot the Data

Create initial visualizations (e.g., line graphs, scatter plots) to get a sense of data trends and distributions.

Inspect the Data (Zoom In)

Explore data in finer detail by zooming into specific time windows or regions to spot patterns, outliers, or errors.

Preprocess by Filtering, if Necessary

Clean the data by filtering out noise or artifacts, especially useful in biological and sensor data.

Transform Data with Machine Learning

Apply machine learning techniques to identify patterns, make predictions, or extract features.

Statistical Analysis

Conduct statistical tests to quantify differences, relationships, or trends, aiding in the interpretation of results.

Best Practices for Effective Coding

Use Meaningful Variable Names
Choose descriptive names for variables so that the purpose of each variable is clear at a glance.
Comment Code
Add comments to explain the purpose of complex or non-obvious sections of code, making it easier to understand later.
Switch Between Code Cells and Text Cells for Notes (in Jupyter Notebooks)
Use markdown cells for notes or explanations, helping organize code and improve readability for yourself and others.
Save Versions of Your Code with Dates in the Filename
Regularly save versions of your code with date-stamped filenames, allowing you to track changes over time and revert if needed.
Use Meaningful File Names and Folder Structures
Organize files and folders logically with descriptive names to easily locate and manage different components of a project.
Maintain Consistency in Code Style
Adhere to a consistent style (e.g., spacing, indentation, and naming conventions) to make code more readable and professional.