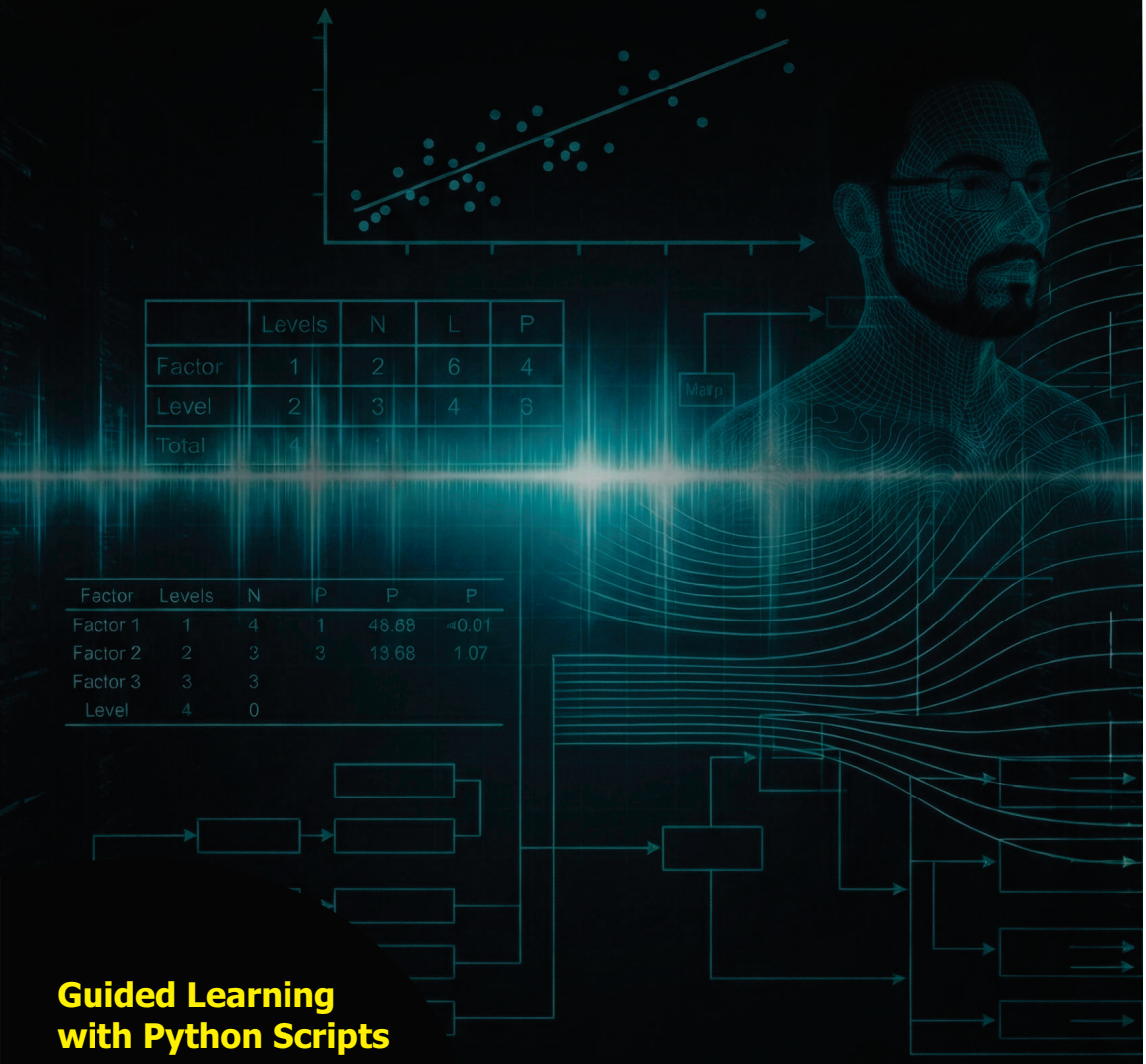
Experimental Design for Artificial Intelligence Engineers



Guided Learning with Python Scripts

Dr. Javier Salas García

Experimental Design for Artificial Intelligence Engineers: Guided Learning with Python Scripts

Author:
Salas-García, Javier

Publication Details:
Publisher: KDP Amazon
ISBN: 9798246721568
Publication Date: February 2026

How to cite this work:
Salas-García, J. (2026). *Experimental Design for Artificial Intelligence Engineers: Guided Learning with Python Scripts.* KDP Amazon.

Printed by Amazon.

This page was intentionally left blank.

Contents

Preface **7**

1 Experimental Design **9**
1.1 Basic concepts of experimental design 9
1.1.1 Identification of Variables 10
1.1.2 Experimental and Sampling Units 11
1.1.3 The Impact of Design on IA Data Quality . . . 12
1.2 Experiment types 13
1.2.1 Comparative and Single-Factor Experiments . 14
1.2.2 Factorial Experiments 15
1.2.3 Field, Laboratory, and Computational Experiments 16
1.3 Experimental and sampling units 17
1.3.1 Defining the Experimental Unit in Intelligence Systems 18
1.3.2 Sampling Units and Data Granularity 18
1.3.3 Pseudoreplication in Machine Learning Datasets 18
1.4 Variation sources 19
1.4.1 Natural and Induced Variation 20
1.4.2 Systematic and Random Variation 20
1.5 Statistical design properties 22
1.5.1 Replication: Precision and Error Estimation . . 23
1.5.2 Randomization: Protecting Against Bias 24
1.5.3 Local Control and Blocking 25
1.6 Randomization 26
1.6.1 Confounding and Temporal Drift 26
1.6.2 Randomization in Automated Data Acquisition 27
1.7 Classification of experiments 28
1.7.1 Screening Designs 29
1.7.2 Characterization and Optimization 29
1.8 Treatment design 30
1.8.1 Selection of Factor Levels 31
1.8.2 Operational Envelope vs. Failure Limits 31

2 Inferential Statistics **33**

2.1 Exploratory data analysis . . . 33
2.1.1 Distributions and Central Tendency . . . 33
2.1.2 Normality and Outlier Detection . . . 34
2.2 Statistical parameters . . . 35
2.2.1 Measures of Center and Spread . . . 36
2.2.2 Shape Parameters: Skewness and Kurtosis . . . 36
2.3 Sampling theory . . . 38
2.3.1 The Law of Large Numbers . . . 39
2.3.2 The Central Limit Theorem (CLT) . . . 40
2.4 Estimation theory . . . 41
2.4.1 Point Estimation and Bias . . . 41
2.4.2 Confidence Intervals: Quantifying Uncertainty 41
2.4.3 Maximum Likelihood Estimation (MLE) . . . 42
2.5 Hypothesis testing . . . 43
2.5.1 The Null and Alternative Hypotheses . . . 44
2.5.2 Decision Errors: Type I and Type II . . . 44
2.5.3 Statistical Power and p-values . . . 44

3 Methods for the Design of Experiments 47
3.1 Regression and correlation theory . . . 47
3.1.1 Linear Regression and Least Squares . . . 47
3.1.2 Diagnostics and Residual Analysis . . . 48
3.2 Single-factor experiments . . . 49
3.2.1 Partitioning Variation: The F-Test . . . 49
3.2.2 Post-hoc Analysis and Multiple Comparisons . 50
3.3 Factorial experiments (two or more factors) . . . 51
3.3.1 Efficiency vs. One-Factor-at-a-Time (OFAT) . 52
3.3.2 The Statistical Model of Interaction . . . 52
3.3.3 Main Effects and Hidden Dynamics . . . 52
3.3.4 Visualizing Interactions: Synergy and Antagonism . . . 53
3.4 2^k Factorial experiments and fractions . . . 54
3.4.1 The Pareto Principle and Factor Sparsity . . . 55
3.4.2 Fractional Factorial Designs: 2^{k-p} . . . 55
3.4.3 Design Resolution and Aliasing Risks . . . 55
3.5 Statistical software . . . 57
3.5.1 Python as an Experimental Platform . . . 57
3.5.2 R and Other Professional Packages . . . 58

4 Application of the Experimental Design 59

4.1 Introduction to Case Studies . . . 59
4.1.1 System 1: Raspberry Pi Temperature Controller 59
4.1.2 System 2: Soft Pneumatic Actuator (SPA) . . 60
4.1.3 System 3: Bubble Flowmeter (Computer Vision) 60
4.1.4 Comparative Dynamics and Sampling Strategy 60
4.2 Identification of variables and Case Study setup 61
4.2.1 System Description and Operational Constraints 62
4.2.2 Identification and Classification of Variables . . 62
4.2.3 Collinearity and Feature Redundancy 63
4.3 Data acquisition and automated experimentation . . . 64
4.3.1 The Communication Bridge: Serial Protocol . . 64
4.3.2 Stabilization vs. Measurement Window 65
4.3.3 Randomized Execution Loop 65
4.4 Selection of experimental theory 66
4.4.1 Knowledge Hierarchy and Information Density 67
4.4.2 Decision Criteria for AI Systems 67
4.4.3 Aliasing and Confounding Trade-offs 68
4.5 Selection of theory and Case Study resolution 69
4.5.1 Execution of a Central Composite Design . . . 70
4.5.2 Visualizing the Optimization Surface 70
4.5.3 Model Validation and Residuals 70
4.6 Interpretation, Evaluation, and AI integration 72
4.6.1 Quantifying Model Fidelity 72
4.6.2 Cross-Hardware Validation and Generalization 73
4.6.3 Safety and Deployment 74

A Activity Schedule and Laboratory Practices 75

B Data Visualization and Statistical Interpretation 77
B.1 Time-Series Plots 77
B.2 Scatter and Strip Plots 77
B.3 Histograms and KDE Plots 77
B.4 Quantile-Quantile (Q-Q) Plots 78
B.5 Boxplots 78
B.6 Violin Plots 79
B.7 Confidence Interval Plots 79
B.8 Residual Plots 79
B.9 Interaction Plots 79
B.10 Pareto Charts of Effects 79
B.11 Response Surface Plots (3D) 80

B.12 System Architecture and Workflow Diagrams 80

Preface

This textbook is designed specifically for undergraduate students of Artificial Intelligence Engineering, aiming to bridge the gap between abstract statistical theory and the physical reality of hardware implementation. In the development of intelligent systems, the quality of training data is a fundamental constraint; therefore, mastering the methodology of experimental design is not an optional skill but a requirement for ensuring the robustness and reliability of AI models. To ground these theoretical concepts, three distinct experimental systems are utilized throughout the book, including a soft pneumatic actuator for spasticity rehabilitation, a Raspberry Pi liquid temperature controller, and a bubble-based air flowmeter for low flows. These systems allow for the study of non-linear mechanical responses, steady-state error, and the conversion of visual signals into quantitative physical data using tools like OpenCV and Python.

The course structure centers on a project-based learning approach where students are expected to select one of these systems as their primary focus for the semester while utilizing the others during laboratory practices. This rotation ensures a comprehensive understanding of different physical phenomena and the unique data acquisition challenges they present. This material is intended to be a flexible resource for the academic community, providing principles of experimental design that are universal and easily adaptable to other laboratory setups or research objectives in the field of intelligent instrumentation. To support this practical learning, all Python scripts used throughout these chapters are available for download at https://electrovigia.net/books. The inclusion of these automated tools provides a blueprint that can be replicated or modified to suit different hardware architectures, ensuring that the engineer moves beyond trial and error toward a foundation of scientific truth.

Dr. Javier Salas García

This page intentionally left blank.

Chapter 1

Experimental Design

1.1 Basic concepts of experimental design

Learning Objective

Analyze the fundamental principles of experimental design by identifying variables and units within the context of artificial intelligence to establish a rigorous framework for data acquisition and model validation.

Data Engineering from Hardware

For an AI engineer, data is the raw fuel of every algorithm. However, not all data is created equal. Understanding experimental design is what separates a «data scraper» from a «data scientist». If the generation of data is not controlled, the resulting model will learn the noise of the hardware rather than the logic of the problem itself.

Experimental design is far more than a simple set of statistical rules; it is the strategic architecture of discovery. In the realm of Artificial Intelligence and hardware integration, it is defined as a systematic approach to purposely change the input variables of a process to observe and identify the reasons for changes in the output response. Imagine the development of the control logic for the Soft Pneumatic Actuator described in this course. It is necessary to know exactly how much pressure the silicone cylinder exerts for every increment in the pump's power. Without a formal design, thousands of points might be collected only to discover later that the ambient temperature or the wear of the material confounded the results, leading to a biased training set that fails in real-world rehabilitation scenarios.

The primary purpose of experimental design in Engineering is to maximize the information gained while minimizing the resources

spent. This is particularly vital when dealing with systems like the Temperature Controller or the Bubble Flowmeter, where each experimental run consumes time and energy. By applying these concepts, it is ensured that the variance observed in the datasets is due to the factors being investigated and not to uncontrolled fluctuations. This rigor allows the construction of models that are not only accurate but also robust against the inherent stochasticity of the physical world.

1.1.1 Identification of Variables

In any experiment, a clear distinction must be made between what is controlled and what is observed. These are the independent and dependent variables. Independent variables, often called factors, are those manipulated to observe their effect on the system. For instance, in a Raspberry Pi Temperature Controller, the independent variable could be the volume of the liquid or the target setpoint. In contrast, the dependent variable—the response—is measured as a result of those manipulations, such as the time required to reach thermal stability or the Root Mean Square Error (RMSE) of the steady-state temperature.

$$Y = f(X_1, X_2, \ldots, X_n) + \epsilon \tag{1.1}$$

In the general model shown above, Y represents the dependent variable, X_i are the independent factors, and ϵ constitutes the experimental error or noise. The goal in AI is often to approximate the function f using machine learning, but it must first be ensured that ϵ is minimized and well-characterized through proper design. As shown in Figure 1.1, which was generated using the script `s1_1_e1.py`, the relationship between the pump's duty cycle and the resulting pressure is rarely a perfect line. This visualization utilizes a scatter plot for the raw observations and a dashed line for the characterization. It is observed that for a duty cycle of 100%, the pressure reaches approximately 12.5 kPa, but the individual points scatter around this value with a standard deviation of 0.4 kPa, representing the intrinsic noise that an AI model must handle.

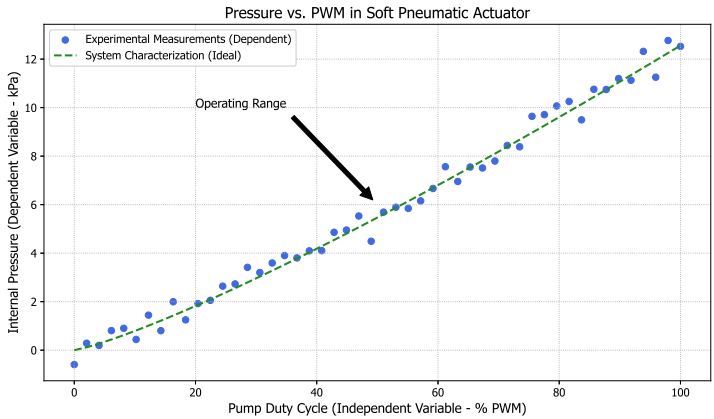


Figure 1.1: Characterization of variables in the Soft Pneumatic Actuator. The independent variable (Duty Cycle) is systematically varied to observe the response in the dependent variable (Internal Pressure), highlighting the natural dispersion that constitutes experimental noise. (Script: s1_1_e1.py)

1.1.2 Experimental and Sampling Units

A common pitfall is confusing the experimental unit with the sampling unit. The experimental unit is the smallest division of material to which a treatment can be applied independently. For example, if three different heating elements are tested in the Liquid Temperature Controller, the entire container of water assigned to one specific heater is the experimental unit. If four different sensors are then placed at different depths within that same container to obtain more data, each of those sensors provides a sampling unit.

It is an error to treat multiple sampling units as if they were independent experimental units. Figure 1.2, generated by the script `s1_1_e2.py`, illustrates this concept by overlapping multiple time-series plots. Each line represents a different sensor (sampling unit). Notice how all three sensors converge to the 55 °C setpoint starting from approximately 60 °C or 61 °C. The shaded grey region highlights the context of the single experimental unit; because the sensors share the same water volume, their thermal trajectories are highly correlated, providing only redundant information about the global treatment rather than independent replications.

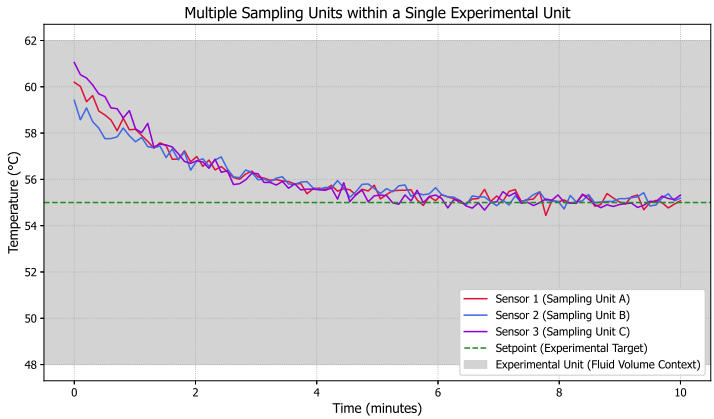


Figure 1.2: Differentiation between sampling and experimental units. The grey area represents the single experimental unit (the thermal environment), while the individual lines represent multiple sampling units (sensors) tracking the convergence to the target setpoint. (Script: s1_1_e2.py)

1.1.3 The Impact of Design on IA Data Quality

The quality of training data determines the upper bound of a model's performance. As seen in Figure 1.3, produced by the script `s1_1_e3.py`, an experiment without proper design results in inconsistent data with overlapping distributions. This figure employs histograms to visualize the density of measurements. In the uncontrolled case (left), the observations for a nominal flow of 3 ml/min are spread between 0 and 6 ml/min, creating a wide and shallow distribution. In contrast, the designed experiment (right) results in a narrow peak centered precisely at 3 ml/min with most observations within a ± 0.5 ml/min range. This reduction in variance significantly simplifies the classification task for any intelligent algorithm.

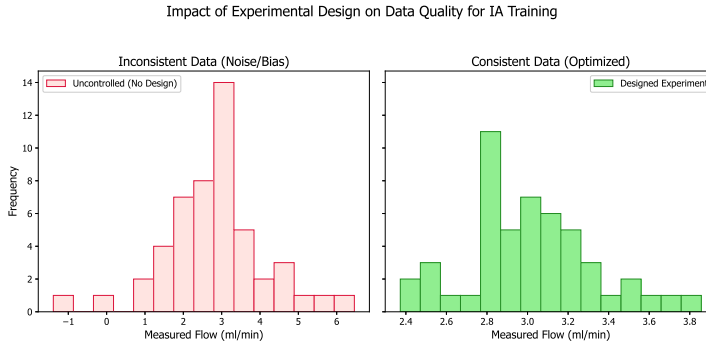


Figure 1.3: Impact of experimental design on data distribution. The comparison shows how a designed experiment reduces variance and eliminates biases, providing a much higher signal-to-noise ratio for training AI models. (Script: s1_1_e3.py)

Computational Challenge: Experiment with the Script

Open the script `s1_1_e1.py` and modify the `noise` parameter in the `generate_pressure_data` function. Increase the standard deviation from 0.4 to 1.5. Observe how the dispersion of points makes it harder to visualize the ideal system behavior. Now, try to decrease the number of points from 50 to 5. Explain why having few points with high noise leads to catastrophic overfitting in a deep learning model.

1.2 Experiment types

Learning Objective

Classify and analyze different types of experimental structures applicable to Artificial Intelligence and electronic systems to select the most appropriate methodology for data characterization and model validation.

Data Engineering from Hardware

Selecting the correct type of experiment is a decision of engineering efficiency. A poorly chosen structure leads to wasted energy and redundant data, whereas a well-planned design isolates the crucial factors that an AI model needs to learn. This classification ensures that the experimental effort is proportional to the complexity of the phenomenon under study.

The landscape of experimentation in Engineering and AI can be broadly categorized according to the structure of the treatments and the nature of the variables involved. A formal taxonomy allows for the identification of the most efficient path to data acquisition, considering the constraints of time, hardware stability, and computational resources.

1.2.1 Comparative and Single-Factor Experiments

Comparative experiments are designed to determine if there is a significant difference between two or more treatments. Figure 1.4, which was generated by the script `s1_2_e2.py`, demonstrates how the steady-state error can be compared across different control strategies. This visualization employs boxplots, which summarize the distribution using five key statistics: the minimum, the first quartile (Q1), the median, the third quartile (Q3), and the maximum. In the figure, the On-Off controller exhibits a median error of approximately 1.2 °C with a wide interquartile range (IQR = Q3 - Q1) spanning from 1.0 °C to 1.5 °C. In contrast, the PID controller shows a significantly lower median error of 0.3 °C with a very narrow IQR, indicating much higher precision. Points outside the whiskers (if any) would represent outliers—observations that fall more than 1.5 times the IQR away from the quartiles.

Single-factor experiments extend this logic by investigating a single independent variable across multiple levels. The resulting data is illustrated in Figure 1.5, generated by the script `s1_2_e1.py`. This figure utilizes violin plots, which provide a more detailed view than traditional boxplots by adding a Kernel Density Estimate (KDE) to the sides. For a pressure level of 70 kPa, the violin width is greatest around 15.5 mm of displacement, indicating that this is the most probable value (the mode). The internal boxplot shows that 50%

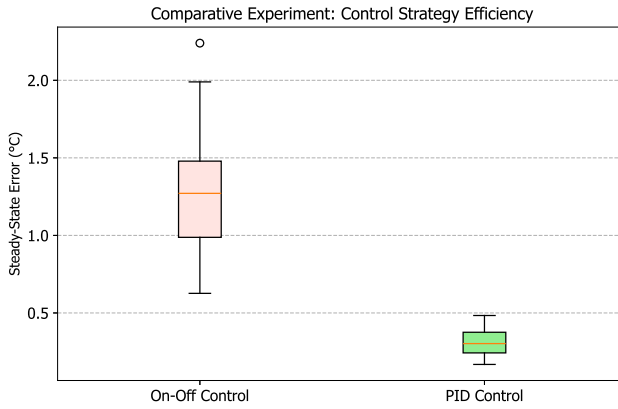


Figure 1.4: Comparative experiment results. The distribution of steady-state error for two different control strategies (On-Off vs. PID) is displayed through boxplots, allowing for a quantitative decision on the most efficient algorithm. (Script: s1_2_e2.py)

of the observations (the IQR) fall between 15.0 mm and 16.0 mm. Unlike the boxplot, the violin plot's curvature reveals if the data has multiple peaks (bimodality), which would suggest a secondary hidden process affecting the hardware.

1.2.2 Factorial Experiments

Factorial experiments allow for the simultaneous investigation of multiple factors and their interactions. Figure 1.6, generated with the script `s1_2_e3.py`, presents these effects using a grouped boxplot arrangement. The figure compares count accuracy across four treatments: Low Flow/Shallow Depth (L-S), High Flow/Shallow Depth (H-S), Low Flow/Deep Depth (L-D), and High Flow/Deep Depth (H-D). It is observed that while changing from L-S (95% accuracy) to H-S results in a drop to 93%, combining High Flow with Deep Depth (H-D) causes a catastrophic drop to 89%. This synergistic effect—where the result of combining factors is worse than the sum of their individual effects—is exactly the type of interaction that an AI model must learn to predict.

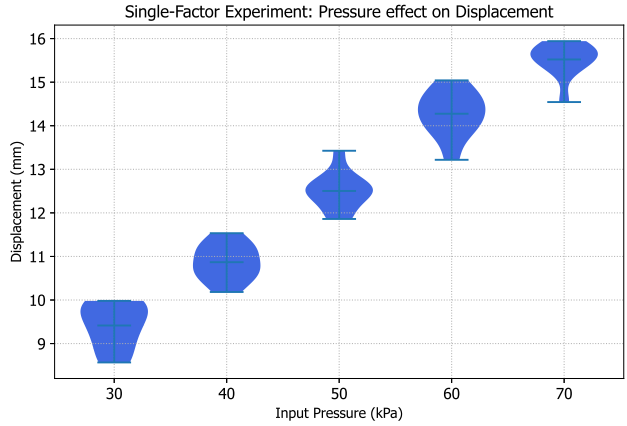


Figure 1.5: Single-factor experiment analysis. The displacement of the soft actuator is tracked across five pressure levels, using violin plots to visualize both the trend and the dispersion of the response. (Script: s1_2_e1.py)

1.2.3 Field, Laboratory, and Computational Experiments

Laboratory experiments are highly controlled, minimizing external disturbances like the ambient light affecting the flowmeter. Field experiments involve testing in the final environment (e.g., a hospital room), where uncontrolled variables become prominent. Finally, computational experiments involve simulations where millions of runs can be executed. Each type serves a specific stage of the development lifecycle, from initial prototyping to final validation.

Computational Challenge: Experiment with the Script

Execute the script `s1_2_e3.py` and observe the factorial results. Modify the `interaction` variable from -1 to -10. Discuss how a strong negative interaction term changes the interpretability of the factors. Consider how an AI model would struggle to generalize if it were trained only on data where factors were varied one at a time, completely ignoring these combined effects.

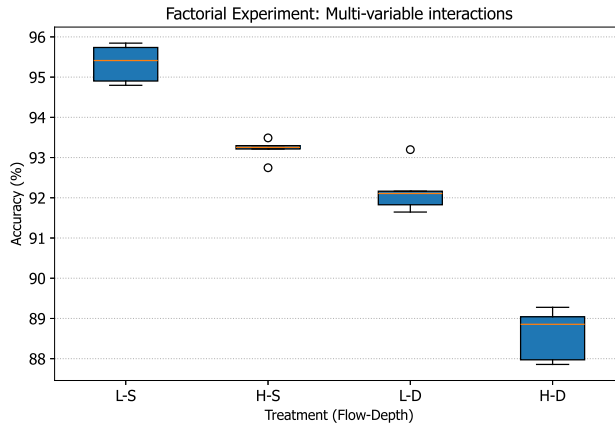


Figure 1.6: Factorial experiment visualization. The accuracy of the flow counting algorithm is evaluated across combinations of flow levels and nozzle depths, highlighting how interactions between factors affect the system performance. (Script: s1_2_e3.py)

1.3 Experimental and sampling units

Learning Objective

Establish a clear distinction between experimental and sampling units within the context of AI datasets and physical sensors to avoid pseudoreplication and ensure the statistical validity of the experimental conclusions.

Data Engineering from Hardware

Confusing a sampling unit with an experimental unit is a common source of inflated confidence in AI models. If a model is trained on multiple measurements taken from the same setup as if they were independent, it will likely fail when deployed on a different setup. Recognizing the true experimental unit is essential for building models that generalize correctly across different physical instances.

In the methodology of experimental research, identifying the smallest unit to which a treatment is applied independently is a critical step.

This smallest division is defined as the experimental unit. In contrast, the sampling unit is the specific part of the experimental unit where a measurement is actually performed.

1.3.1 Defining the Experimental Unit in Intelligence Systems

The experimental unit is the entity that receives a treatment randomly. In AI Engineering, failing to recognize this leads to results that do not replicate. For example, if a model captures characteristics unique to a single batch of material rather than the intended treatment, the conclusions will be biased.

1.3.2 Sampling Units and Data Granularity

A sampling unit is a discrete observation taken from within an experimental unit. Figure 1.7, generated by the script `s1_3_e1.py`, illustrates this relationship through time-series plots. Each of the three independent lines represents a measurement from a different thermocouple (sampling unit) within the same water container (experimental unit). Notice that while each sensor has its own noise—fluctuating by about ± 0.4 °C—all three follow the exact same heating curve from 20 °C to 75 °C. For a statistical test, this does not represent three independent replications of the experiment, but rather a single replication observed three times.

1.3.3 Pseudoreplication in Machine Learning Datasets

When sampling units are treated as independent experimental units, a phenomenon known as pseudoreplication occurs. This leads to an artificial reduction in the estimated experimental error. In computer vision, taking 1,000 frames from a single 1-minute video does not provide 1,000 independent data points. A model trained on such data will demonstrate high accuracy on the validation set but will degrade significantly when encountering a different setup, as it has learned the specific biases of that single experimental unit rather than the underlying physics.

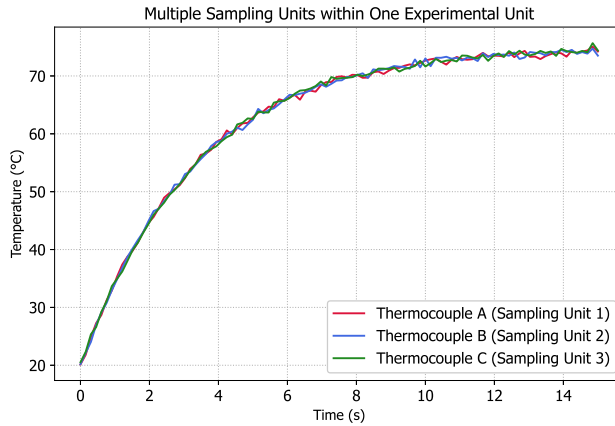


Figure 1.7: Observation of multiple sampling units within a single experimental unit. The shared trend highlights the lack of statistical independence between sensors monitoring the same physical process. (Script: s1_3_e1.py)

Computational Challenge: Experiment with the Script

Examine the script `s1_3_e1.py` and observe the trajectories of the three sampling units. Modify the `base_process` to simulate a faster heating rate and rerun the simulation. Note how all sampling units follow the same change. Discuss why an AI model, if trained only on these three sensors without other independent runs, would be unable to distinguish between sensor noise and a true change in the physical heating process.

1.4 Variation sources

Learning Objective

Identify, classify, and quantify the different sources of variation that affect experimental data to implement control strategies that minimize their impact on the performance of Artificial Intelligence models.

Data Engineering from Hardware

A dataset without an understanding of its variation is merely a collection of numbers. For an AI system, variation is the primary enemy of accuracy. If a model cannot distinguish between the real signal and the noise, it will eventually overfit to the random fluctuations of the hardware, leading to failures in generalization.

In any experimental process, the observed response will fluctuate even when the input factors are kept seemingly constant. This fluctuation originates from multiple sources that must be managed. In the context of the physical systems utilized in this course, variation is categorized into four primary types: natural, induced, systematic, and random.

1.4.1 Natural and Induced Variation

Natural variation is the intrinsic fluctuation present in any physical process. Figure 1.8, generated by the script `s1_4_e1.py`, illustrates this by showing a time-series plot of a temperature reading meant to be stable at 55 °C. Observe the low-frequency drift—the slow wave oscillating within a ±0.5 °C range—and the high-frequency «jitter» caused by electronic noise. This visualization is essential for an engineer to determine the noise floor of the instrument.

Induced variation is the change in the response that is purposely created by manipulating factors. Figure 1.9, produced by the script `s1_4_e2.py`, utilizes a strip plot. A strip plot is a 1D scatter plot where points are «jittered» (slightly shifted horizontally) to avoid overlap. The figure shows three distinct clusters corresponding to 20%, 50%, and 80% PWM. While the induced variation (the vertical distance between clusters) shifts the pressure from 7 kPa to 13 kPa, each cluster has its own internal spread of 0.4 kPa. This allows for a visual comparison: if the clusters were touching, the induced effect would be buried in the noise.

1.4.2 Systematic and Random Variation

Systematic variation represents a consistent bias, while random variation is unpredictable. Figure 1.10, generated by the script `s1_4_e3.py`, compares these using Kernel Density Estimate (KDE) plots. A KDE plot represents the probability density function—the

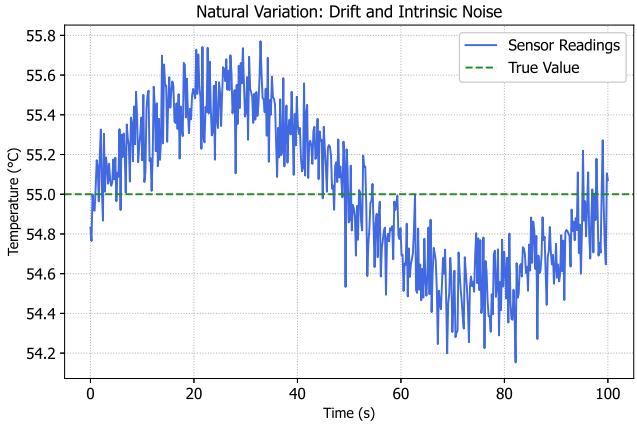


Figure 1.8: Observation of natural variation. The time-series plot reveals environmental drift and intrinsic sensor noise, which constitute the baseline error of the system. (Script: s1_4_e1.py)

peak corresponds to the most frequent value (the mode). It is observed that the calibrated sensor (green) is perfectly centered around the theoretical value of 3.5 ml/min. In contrast, the uncalibrated sensor (red) shows a distribution of the same shape but shifted to the right by 0.8 ml/min. This shift is the systematic bias. An AI model trained on biased data will consistently over-predict the flow, potentially leading to critical failures in control logic.

Computational Challenge: Experiment with the Script

Execute the script `s1_4_e3.py` and observe the bias in the uncalibrated sensor. Modify the `systematic_error` variable from 0.8 to 0.0. Rerun the script and verify that the red distribution now overlaps with the green one.

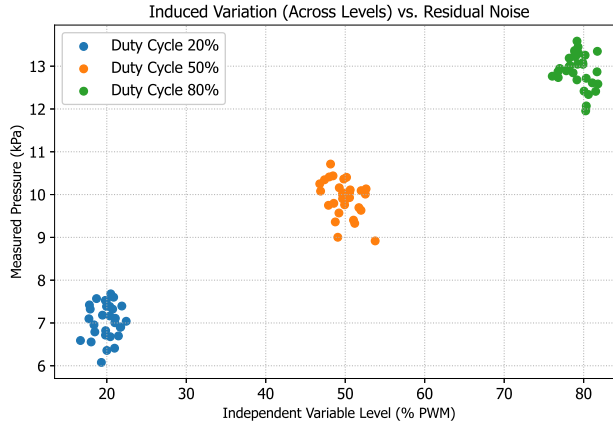


Figure 1.9: Visualizing induced variation vs. residual noise. The strip plot arrangement allows for the direct observation of how changing the input compares to the unavoidable random fluctuations at each level. (Script: s1_4_e2.py)

1.5 Statistical design properties

Learning Objective

Evaluate the fundamental properties of a statistical design, including replication, randomization, and blocking, to ensure that the experimental results provide unbiased and precise estimates of the effects of interest in AI systems.

Data Engineering from Hardware

Mathematical beauty in an AI model is useless if the data used to train it lacks statistical validity. Properties like replication and randomization are the safeguards that prevent an engineer from mistaken noise for patterns. Without these properties, a design is not an experiment but a series of anecdotal observations that cannot be generalized to new hardware or environments.

A well-constructed experimental design must satisfy specific criteria to be considered statistically sound. These properties ensure that

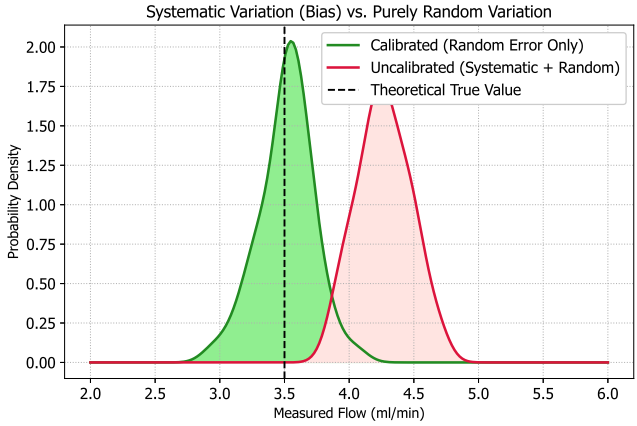


Figure 1.10: Comparison of systematic and random variation. The KDE plots visualize how a calibration error results in a shift of the entire probability density. (Script: s1_4_e3.py)

the estimates of the effects (such as the impact of pressure on an actuator) are both valid and as precise as possible. For an AI engineer, these properties translate directly into data quality: a dataset from a design that satisfies these properties will have a clear, separable signal, whereas a poorly designed one will suffer from confounding and high variance. The three pillars of experimental design are replication, randomization, and local control (blocking).

1.5.1 Replication: Precision and Error Estimation

Replication is the independent repetition of the basic experiment. It is important to distinguish it from repeated measurements (sampling units). For the Soft Pneumatic Actuator, true replication involves manufacturing multiple actuators and testing each one. If only one actuator is tested 100 times, the results represent the behavior of that specific unit, not the general behavior of the silicone formula. Replication allows for the estimation of the experimental error, providing a yardstick to determine if the changes observed in the response are statistically significant.

Figure 1.11, generated by the script `s1_5_e1.py`, illustrates how the standard error of the mean decreases as the number of independent replications increases. This visualization uses a line plot with dy-

namic confidence intervals shaded in green. Notice that with only 2 replications, the uncertainty (the shaded area) is over ±2.0 units, while after 10 replications, it stabilizes below ±0.5 units. In AI, this implies that more replications provide a more stable target for loss function convergence.

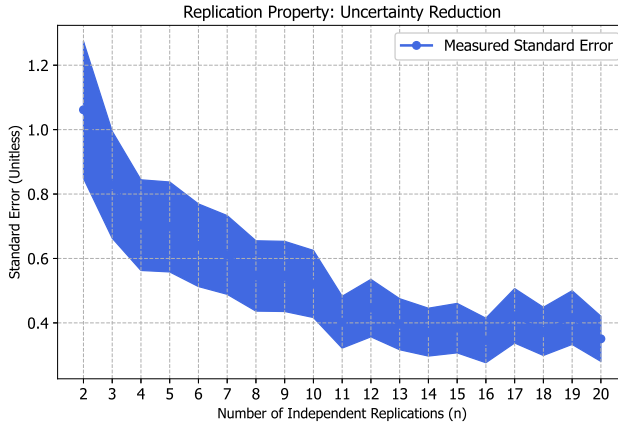


Figure 1.11: Impact of replication on the precision of the estimated effect. The standard error of the mean decreases as more independent replications are added, narrowing the confidence interval around the true system response. (Script: s1_5_e1.py)

1.5.2 Randomization: Protecting Against Bias

Randomization is the practice of assigning treatments to experimental units in a random order. It serves as an insurance policy against confounding variables that the researcher might not even be aware of, such as the gradual warming of the laboratory or the slow depletion of a battery. If a researcher tests all low-pressure levels in the morning and all high-pressure levels in the afternoon, any observed difference might be due to the time of day rather than the pressure itself.

By randomizing the order of experiments, these time-dependent effects are distributed across all treatment levels, converting potential systematic bias into random noise that can be handled statistically. This is particularly crucial for the Raspberry Pi Temperature Controller, where internal thermal components can heat up over several runs, creating a hidden trend in the data.

1.5.3 Local Control and Blocking

Blocking is used to reduce or eliminate the variability arising from known but irrelevant factors (nuisance factors). If three different ESP32 boards are used to collect data, the slight differences in their ADC (Analog-to-Digital Converter) calibration could introduce unwanted variation. By treating each board as a «block» and ensuring that all pressure levels are tested on each board, the variation between boards can be mathematically isolated and removed from the experimental error.

Figure 1.12, produced by the script `s1_5_e2.py`, compares an unblocked design with a blocked one using side-by-side boxplots. The unblocked data (left) shows a wide spread because it mixes the variations of different hardware units. The blocked data (right) shows smaller boxes for each unit, indicating that after accounting for the block effect, the precision of the treatment comparison is significantly improved.

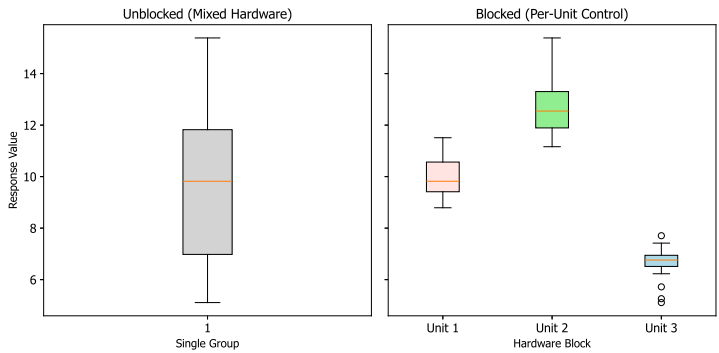


Figure 1.12: Improvement in precision through blocking. Comparing the spread of data when hardware variation is ignored vs. when it is controlled using a blocked design. (Script: s1_5_e2.py)

Computational Challenge: Experiment with the Script

Execute script `s1_5_e1.py` and observe how the error stabilizes. Change the `population_std` from 1.0 to 5.0. How many more replications are needed to reach the same level of precision? Discuss why high-noise hardware (like uncalibrated sensors) significantly increases the cost of data acquisition for AI training.

1.6 Randomization

Learning Objective

Implement randomization strategies in the execution of experiments to eliminate systematic bias and ensure that uncontrollable environmental factors are distributed as random noise across all treatment levels.

Data Engineering from Hardware

Randomization is the insurance policy of a data scientist. In many hardware systems, physical properties change over time—components heat up, sensors drift, and materials fatigue. If an experiment is performed in a fixed order, these temporal changes will be confounded with the factors of interest, leading the AI to learn a pattern that is actually just the passage of time.

Randomization consists of determining the sequence in which experimental runs are performed through a random process, such as a random number generator. Instead of testing all low-input levels followed by all high-input levels, the levels are shuffled. This practice serves two fundamental purposes: it validates the assumption of independent and identically distributed (i.i.d.) errors required for most statistical tests, and it protects the experiment against confounding variables—factors that change during the experiment but are not the primary focus of the study.

1.6.1 Confounding and Temporal Drift

A common example of confounding in AI Engineering occurs when testing the Soft Pneumatic Actuator. As the pump operates, its

internal temperature increases, which slightly changes its efficiency. If an engineer tests 20 kPa, then 40 kPa, and finally 80 kPa, the 80 kPa measurements will coincide with the hottest state of the pump. The observed increase in pressure might be a combination of the power increment and the thermal expansion of the internal valves.

Figure 1.13, generated by the script `s1_6_e1.py`, illustrates this danger. The left plot shows a sequential experiment where a natural drift of 5 mm in displacement (due to material fatigue or sensor warming) is added over time. Because the pressure levels are also increasing, the slope of the dashed red line (the biased fit) is steeper than the real physical relationship. At 80 kPa, the displacement appears to be 12.5 mm, whereas it should be near 8.5 mm. On the right, the same drift is distributed across all levels using randomization. Although the points are more dispersed, the blue dashed line (the unbiased fit) correctly captures the true physical slope, as the drift now contributes only to the random noise rather than the systematic trend.

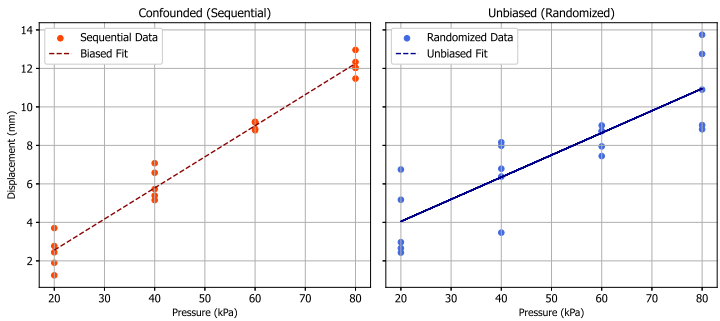


Figure 1.13: Comparison between sequential and randomized experimental orders. Sequential execution confounding the factors with temporal drift, leading to a biased model, while randomization distributes the drift as random error. (Script: s1_6_e1.py)

1.6.2 Randomization in Automated Data Acquisition

In professional laboratory setups, randomization is implemented directly in the control software. Rather than the user manually choosing the next point, the Python script generates a shuffled list of treatments. For the Raspberry Pi Temperature Controller, randomizing the setpoint order prevents the latent heat of the thermal mass from

creating a systematic bias in the steady-state error measurements. By forcing the system to jump between high and low temperatures in a non-linear sequence, the resulting dataset provides a robust representation of the controller's performance across its entire operational envelope, regardless of the starting thermal state.

Computational Challenge: Experiment with the Script

Execute the script `s1_6_e1.py` and observe the difference in slopes between the biased and unbiased fits. Modify the `drift` magnitude from 5 to 10 in the script. Notice how the sequential experiment becomes even more deceptive. Discuss how an AI model trained on the sequential data would fail when the pump is cold, as it would over-predict the displacement based on the biased slope it learned.

1.7 Classification of experiments

Learning Objective

Categorize experimental designs based on their objectives—screening, characterization, or optimization—to select the most efficient structure for the different stages of an AI system development lifecycle.

Data Engineering from Hardware

Experimentation is a finite resource. A common mistake is to perform a detailed optimization experiment on a factor that wasn't even significant. By classifying the experiment correctly, an engineer ensures that the complexity of the data acquisition matches the current knowledge level of the system.

Experiments are classified not only by their mathematical structure but primarily by their intent. In the development of intelligent electronic systems, an experiment usually falls into one of three categories: screening (identifying which factors matter), characterization (modeling how they behave), or optimization (finding their best values). Selecting the wrong class leads to either insufficient data to make decisions or an excessive amount of data for a trivial problem.

1.7.1 Screening Designs

Screening is performed during the initial stages of a project when many potential factors are identified, but only a few are expected to have a dominant effect (the Pareto principle). For the Bubble Flowmeter, candidate factors might include the room lighting, the distance of the camera, the bubble size, and the video resolution. A screening design uses only two widely spaced levels (e.g., Low and High) for each factor to quickly eliminate those that do not contribute significantly to the counting error.

As shown in Figure 1.14 (left), generated by `s1_7_e1.py`, a screening design focuses on the slope between two extreme points. If the response changes from 5 units to 12 units when varying the distance from 2 cm to 8 cm, the factor is considered significant and is kept for the next stage.

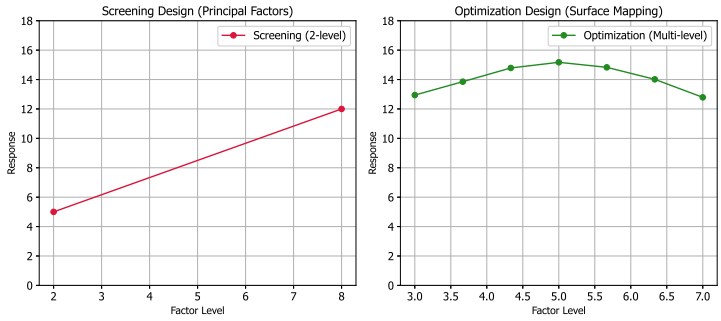


Figure 1.14: Classification of experimental intent. Screening designs (left) use extreme levels to identify significant factors, while Optimization designs (right) use multiple intermediate levels to map the response surface and find the peak performance. (Script: s1_7_e1.py)

1.7.2 Characterization and Optimization

Once the critical factors are identified, the engineer moves to characterization. Here, the goal is to map the functional relationship $Y = f(X)$ in detail. If the relationship is expected to be non-linear, more than two levels are required. Finally, optimization involves finding the combination of factors that yields the best performance—such as the set of control parameters that minimizes the settling time in the Temperature Controller.

Figure 1.14 (right) illustrates an optimization design. Unlike the two points of the screening stage, seven levels are tested between the values of 3 and 7. The data reveals a clear curvature, centered around a peak response of approximately 15 units at a factor level of 5. For an AI model, this type of data is essential for learning the local gradients and identifying the global extrema of the hardware's performance.

Computational Challenge: Experiment with the Script

Open the script `s1_7_e1.py` and look at the `opt_y` calculation. Change the coefficient of the quadratic term from -0.5 to -0.05. Rerun the script. Notice how the optimization peak becomes much harder to identify amidst the noise. Explain why a weak relationship requires many more levels and replications to be captured correctly by a neural network.

1.8 Treatment design

Learning Objective

Define the levels and ranges of the independent factors (treatments) based on physical knowledge of the system and experimental objectives to maximize the information density of the captured data.

Data Engineering from Hardware

A model is only as good as the range of data it has seen. In AI, "interpolation is safe, but extrapolation is dangerous." Choosing treatment levels that span the entire operational envelope of the hardware ensures that the resulting model remains valid in real-world conditions where the inputs are not always ideal.

Treatment design involves deciding which specific values of the independent variables will be tested. This decision includes defining the range (the difference between the minimum and maximum levels) and the granularity (the number of levels within that range). In the context of electronic systems, this is often guided by the specifications

of the sensors and actuators—for instance, the 0 to 100% range of a PWM signal or the 0 to 5V range of an ADC.

1.8.1 Selection of Factor Levels

The choice of levels must be strategic. If the levels are too close together, they might not provide enough variation to distinguish the signal from the noise. If they are too far apart, a researcher might miss important non-linearities in the middle of the range. For the Soft Pneumatic Actuator, if only 0% and 10% duty cycles are tested, the behavior will appear perfectly linear, ignoring the saturation effects that occur above 70%.

Figure 1.15, generated by the script `s1_8_e1.py`, demonstrates the impact of this choice. The black dashed line represents the true physical law—a saturation curve. When levels are clustered only in the linear region (red dots: 10, 20, 30 kPa), the captured data suggests a straight line that would lead to a catastrophic error if used to predict displacement at 90 kPa. In contrast, the expert selection (green diamonds: 10, 40, 90 kPa) correctly captures both the initial growth and the final saturation at approximately 18 mm. This spread of data points is essential for training an AI that can handle the full non-linear dynamics of the system.

1.8.2 Operational Envelope vs. Failure Limits

The experimental range should cover the entire expected operational envelope but must also include the edges of failure. For the Temperature Controller, testing only at 37 °C (the setpoint) is insufficient. The treatments must include extreme cases, such as the maximum volume of liquid or the maximum power of the heater, to evaluate the system's stability limit. For an AI model, data on the "edge" of correct performance is invaluable for building robust decision boundaries that can detect and handle hardware failures before they become critical.

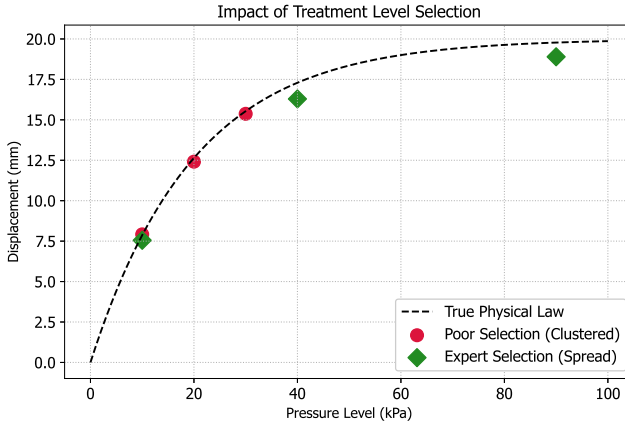


Figure 1.15: Strategic selection of treatment levels. Clustered points (red) fail to capture the non-linear saturation of the system, while spread-out levels (green) provide enough information to model the entire operational range. (Script: s1_8_e1.py)

Computational Challenge: Experiment with the Script

Execute the script `s1_8_e1.py`. Modify the `le_x` variable (Expert Selection) to `[1, 5, 10]`. Rerun the script and observe how even the expert selection becomes poor if it only focuses on a small fraction of the range. Why is it crucial for an AI model to have "anchor points" at the extremes of the operational range?

Chapter 2

Inferential Statistics

2.1 Exploratory data analysis

Learning Objective

Perform a comprehensive exploratory analysis of experimental data using descriptive statistics and visualization techniques to detect outliers, assess normality, and identify patterns before applying formal inferential models.

Data Engineering from Hardware

In AI Engineering, jumping straight to training a model without performing Exploratory Data Analysis (EDA) is a recipe for failure. EDA is the process of «interviewing» the data to discover its biases and limitations. If the data is skewed or contains outliers due to sensor malfunction, the AI model will incorporate these errors as part of the learned logic, leading to poor generalization in real-world scenarios.

Exploratory Data Analysis (EDA) is the essential first step in the analytical pipeline. It involves summarizing the main characteristics of a dataset, often with visual methods, without assuming any underlying statistical model. For an engineer working with physical systems like the Soft Pneumatic Actuator or the Temperature Controller, EDA serves to validate that the data acquisition process was successful and that the measurements align with physical expectations.

2.1.1 Distributions and Central Tendency

The first task in EDA is to visualize the distribution of the response variable. This reveals the central tendency, the spread, and the presence of skewness. Figure 2.1, generated by the script `s2_1_e1.py`, shows the steady-state error of a temperature control system. Notice

that the histogram is not perfectly symmetrical; it exhibits a positive skew, where the tail extends toward the right.

Statistical markers help quantify this observation. In Figure 2.1, the median (orange line) is located at 0.53 °C, while the mean (crimson dashed line) is pulled higher to 0.60 °C by the extreme values in the tail. For an AI engineer, this difference is crucial: if the mean and median are far apart, the data is likely non-normal, and standard loss functions like Mean Squared Error (MSE) might be overly sensitive to these extreme values, potentially biasing the control model.

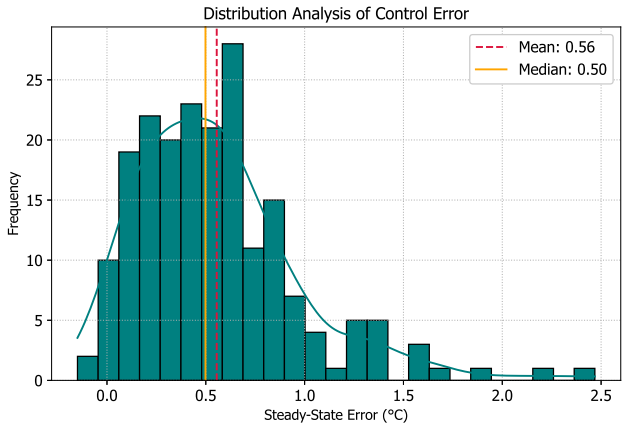


Figure 2.1: Analysis of control error distribution. The histogram and KDE reveal a positive skew where the mean (0.60) is higher than the median (0.53), suggesting the presence of high-error outliers in the control cycle. (Script: s2_1_e1.py)

2.1.2 Normality and Outlier Detection

Many statistical techniques and machine learning algorithms assume that the data follows a Normal (Gaussian) distribution. Assessing this assumption is done through Quantile-Quantile (Q-Q) plots. A Q-Q plot compares the quantiles of the experimental data against the quantiles of a theoretical normal distribution. If the data is normal, the points fall along a straight 45-degree line.

Figure 2.2, produced by the script `s2_1_e2.py`, compares two scenarios. On the left, the points follow the line closely, indicating that the noise in the sensor is purely Gaussian. On the right, the points «curv»

away from the line at the extremes (the tails). This signifies «heavy tails», typically caused by outliers—perhaps due to EMP interference in the ADC readings of the ESP32. Detecting these deviations during EDA allows the engineer to implement filtering pre-processing steps, such as a median filter, before the data is used for model training.

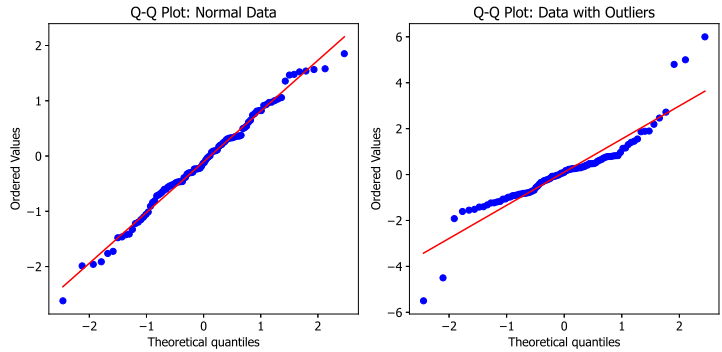


Figure 2.2: Normality check using Q-Q plots. Deviation from the reference line at the extremes indicates the presence of outliers or heavy-tailed distributions that could negatively impact model convergence. (Script: s2_1_e2.py)

Computational Challenge: Experiment with the Script

Execute the script `s2_1_e1.py` and observe the skewness. Modify the `sigma` parameter in the `lognormal` function from 0.4 to 0.8. Rerun the script and note how the mean and median drift further apart. How would this increased skewness affect the training of a linear regression model? Would the model over-predict or under-predict the average error?

2.2 Statistical parameters

Learning Objective

Define and calculate critical statistical parameters—including mean, variance, skewness, and kurtosis—to quantify the performance metrics of hardware components and integrate these parameters into AI validation protocols.

Data Engineering from Hardware

Deep Learning models are essentially statistical parameter estimators on a massive scale. To understand how an AI learns, an engineer must first understand what a parameter is. A simple change in the variance of a sensor can be the difference between a model that generalizes and one that overfits to experimental noise.

Statistical parameters are numerical summaries that describe the behavior of a population or a sample. In experimental design for AI, these parameters are not just abstract numbers; they represent the physical reality of the hardware. For instance, the mean pressure in the Soft Pneumatic Actuator tells us about its strength, while the variance tells us about its reliability.

2.2.1 Measures of Center and Spread

The mean (μ) provides the central value of the distribution, while the variance (σ^2) and standard deviation (σ) quantify the dispersion. Figure 2.3, generated by the script `s2_2_e1.py`, illustrates how two systems with the same mean behavior can be of vastly different quality.

The green distribution has a low standard deviation ($\sigma = 0.5$). For a target reading of 0.0, most observations fall between -1.0 and 1.0. In contrast, the red distribution has a high standard deviation ($\sigma = 1.5$), spreading the observations between -4.5 and 4.5. For an AI model, the red system is much harder to control: the uncertainty is three times higher, which significantly increases the risk of the algorithm making an incorrect decision based on a noisy input. High variance in training data reduces the signal-to-noise ratio, often requiring much larger datasets to achieve the same accuracy as a low-variance system.

2.2.2 Shape Parameters: Skewness and Kurtosis

Beyond the center and spread, we must characterize the shape of the data. Skewness measures the asymmetry of the distribution. A positive skew indicates a long tail to the right, often seen in latency measurements where a few runs take much longer than the average.

Kurtosis measures the «tailedness» of the distribution. Figure 2.4, produced by the script `s2_2_e2.py`, compares different degrees of

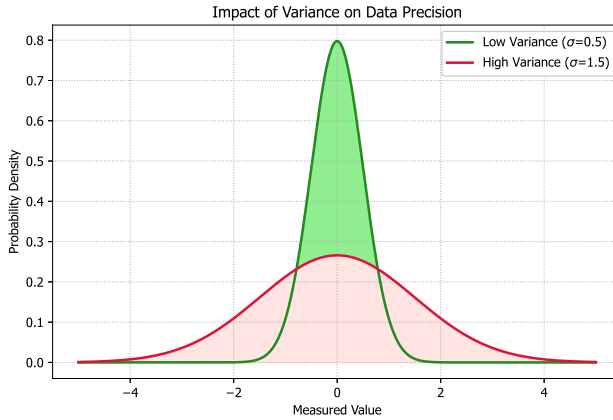


Figure 2.3: Comparison of variance in experimental readings. Both systems share the same mean, but the high-variance distribution (red) exhibits three times more dispersion, representing a less precise hardware environment. (Script: s2_2_e1.py)

kurtosis. The leptokurtic distribution (purple) has a sharp peak and heavy tails, meaning most values are very close to the mean, but extreme outliers are more frequent than in a normal distribution. The platykurtic distribution (orange) is flat, meaning the data is spread more uniformly across the range. For an AI engineer, high kurtosis is a warning sign: it indicates that while the system is usually stable, it produces infrequent but extreme errors that could trigger a safety shutdown in an autonomous rehabilitator.

Computational Challenge: Experiment with the Script

Open the script `s2_2_e1.py` and modify the high variance standard deviation from 1.5 to 5.0. Observe how the red curve almost disappears into a flat line. Discuss why an AI model would struggle to find any "pattern" in such a distribution. If this were a sensor for a medical robot, what would be the safety implications of such high variance?

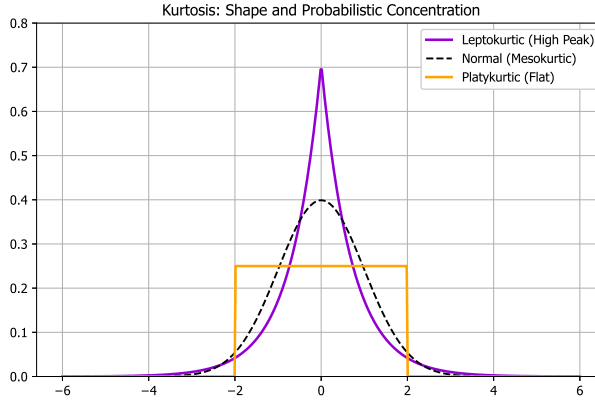


Figure 2.4: Analysis of kurtosis and distribution shape. The sharp peak of the leptokurtic case (purple) indicates highly consistent readings but with a higher probability of extreme outliers in the tails compared to the normal case (black). (Script: s2_2_e2.py)

2.3 Sampling theory

Learning Objective

Apply sampling theory to experimental data acquisition, understanding the Law of Large Numbers and the Central Limit Theorem to determine representative sample sizes for AI model validation.

Data Engineering from Hardware

We never have access to the infinite truth of a physical system; we only have a sample. The bridge between our finite measurements and the global behavior of the hardware is sampling theory. If an AI is trained on a small, non-representative sample, it will fail to capture the underlying physics, regardless of how complex the neural architecture is.

Sampling theory provides the mathematical justification for why we can draw conclusions about a large population (e.g., all Soft Pneumatic Actuators manufactured) based on a limited number of

observations. In AI Engineering, this theory is the foundation for defining training, validation, and test sets.

2.3.1 The Law of Large Numbers

The Law of Large Numbers (LLN) states that as the number of independent samples increases, the sample mean ($\bar{x}$) converges to the true population mean (μ). For the Raspberry Pi Temperature Controller, if we only take 5 measurements, the average steady-state error might be significantly biased by a single outlier. However, if we take 1,000 measurements, the random fluctuations cancel out.

Figure 2.5, generated by the script `s2_3_e2.py`, demonstrates this property. The navy line shows the running average of a simulated measurement. In the first 50 samples, the average oscillates wildly between 9.5 and 10.5. As the sample size reaches 1,000, the average stabilizes precisely at the true value of 10.0 (red dashed line). This implies that a larger dataset is always more reliable, but also reveals a point of diminishing returns after approximately 400 samples, where the gain in precision becomes minimal compared to the cost of acquisition.

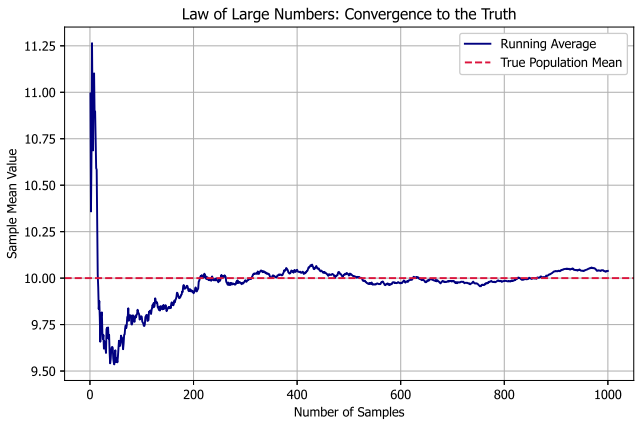


Figure 2.5: Law of Large Numbers demonstration. The running average of the sample converges to the population mean as the number of observations increases, reducing the impact of individual measurement errors. (Script: s2_3_e2.py)

2.3.2 The Central Limit Theorem (CLT)

The Central Limit Theorem is perhaps the most powerful tool in statistics. It states that the distribution of the sample mean follows a Normal distribution as the sample size increases, regardless of the shape of the original population distribution. This is why many AI models can assume Gaussian errors even when the underlying sensor noise is not strictly Normal.

Figure 2.6, produced by the script `s2_3_e1.py`, shows this convergence starting from a highly non-normal (exponential) population. With a sample size of $n = 2$, the distribution of the mean is still heavily skewed. However, by the time we reach $n = 30$, the distribution is clearly bell-shaped and centered around the population mean. This $n = 30$ is often cited as the «magic number» in experimental design; beyond this sample size, the reliability of the statistical estimates increases significantly, providing a stable foundation for the AI to learn the system's core dynamics.

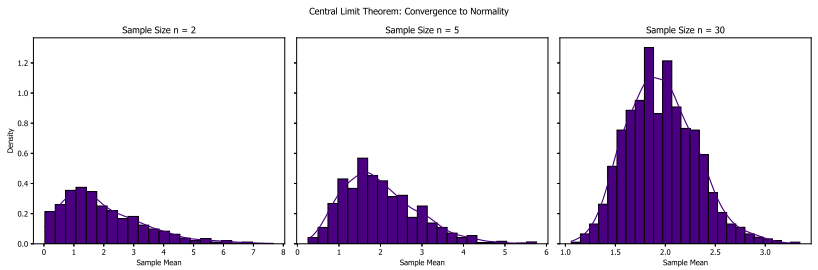


Figure 2.6: Central Limit Theorem effect. Even with non-normal population data, the distribution of the sample mean becomes normal as the sample size n increases, enabling the use of standard statistical inference. (Script: s2_3_e1.py)

Computational Challenge: Experiment with the Script

Execute the script `s2_3_e1.py` and observe the three histograms. Modify the population type in `demonstrate_clt` from `exponential` to `uniform`. Rerun the script. Does the convergence to normality happen faster or slower than with the exponential case? Why is it safer to train an AI on the "mean of batches" rather than on individual, extremely noisy raw readings?

2.4 Estimation theory

Learning Objective

Evaluate and apply point and interval estimation methods, specifically confidence intervals for the mean and variance, to provide a probabilistic range for the performance parameters of an Artificial Intelligence system.

Data Engineering from Hardware

A point estimate (like the average accuracy) is a single number that is almost certainly wrong. Estimation theory allows us to move from a single, likely incorrect number to a range where we have a calculated confidence of being correct. In AI validation, it is not enough to say "the model is 95% accurate"; we must say "we are 95% confident the accuracy is between 93% and 97%."

Estimation theory deals with the construction of estimators—rules for calculating an estimate of a population property based on observed data. There are two primary types of estimation: point estimation (a single value) and interval estimation (a range of values).

2.4.1 Point Estimation and Bias

A point estimate uses sample data to calculate a single value which is to serve as a «best guess» for an unknown population parameter. For the Bubble Flowmeter, the sample mean of bubble counts over 10 minutes is a point estimate of the true flow rate. However, an estimator is only as good as its properties, primarily its bias. An unbiased estimator, such as the sample mean $\bar{X}$, has an expected value equal to the true population parameter μ. If the ADC of the ESP32 is not calibrated, the point estimate of the pressure will be biased, shifting all subsequent AI training toward an incorrect physical reality.

2.4.2 Confidence Intervals: Quantifying Uncertainty

A Confidence Interval (CI) provides a range of values that is likely to contain the true population parameter. The confidence level (e.g.,

95%) represents the frequency with which the interval would contain the true parameter if the experiment were repeated many times.

Figure 2.7, generated by the script `s2_4_e1.py`, visualizes this concept. Fifty different samples were taken from a population with a true mean of 50.0. For each sample, a 95% Confidence Interval was calculated. It is observed that most intervals (in green) contain the black dashed line (the truth). However, approximately 1 out of every 20 intervals (in red, such as sample number 41 or 12) fails to capture the true mean despite following the correct statistical procedure. This is the inherent probabilistic risk of experimentation. For an AI engineer, the width of these bars (the margin of error) is a direct measure of the dataset's uncertainty; a wide bar indicates that the AI is being trained on highly unstable data.

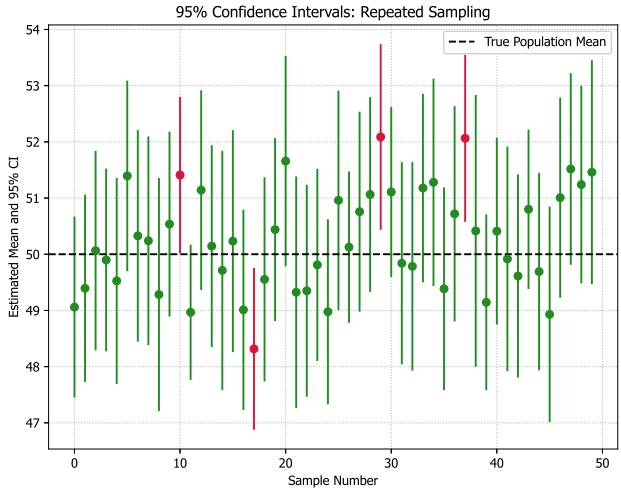


Figure 2.7: Visualizing 95% Confidence Intervals. While the majority of intervals contain the true population mean (50.0), the probabilistic nature of sampling guarantees that a small percentage will fail to do so, highlighting the risk of drawing conclusions from single experiments. (Script: s2_4_e1.py)

2.4.3 Maximum Likelihood Estimation (MLE)

Maximum Likelihood Estimation is the backbone of most deep learning training algorithms (like cross-entropy loss). It involves finding

the parameter values that maximize the likelihood of making the observations given the parameters. When we train a neural network to predict the displacement of the Soft Pneumatic Actuator, we are essentially finding the weights that make the training data most probable. Understanding MLE allows the engineer to design custom loss functions that are tailored to the specific noise characteristics of the hardware sensors.

Computational Challenge: Experiment with the Script

Open the script `s2_4_e1.py` and change the confidence level from 95% (using 1.96) to 99% (using 2.58). Rerun the script. Notice how the error bars become much wider. Why does increasing our confidence require a wider interval? If we needed to reduce the margin of error (the width of the bars) without decreasing our confidence, what change would we need to make to the experimental process (the sample size n)?

2.5 Hypothesis testing

Learning Objective

Formulate, execute, and interpret statistical hypothesis tests to validate performance claims of AI hardware systems and decide whether observed differences are significant or the result of random variation.

Data Engineering from Hardware

In AI Engineering, we are constantly making decisions: Is the new control logic faster? Is the second sensor more accurate? Hypothesis testing is the formal math of decision-making. It prevents us from being fooled by random luck. A p-value is essentially the probability that what we are seeing is just a fluke. If that probability is low (typically < 0.05), then we have discovered something real.

Hypothesis testing is a formal procedure for investigating our ideas about the world using statistics. It is most often used to test specific

predictions that arise from theories or observed patterns in experimental data.

2.5.1 The Null and Alternative Hypotheses

The Null Hypothesis (H_0) is a statement of "no effect" or "no difference." For example, if comparing two PID tunings for the Temperature Controller, H_0 states that both tunings result in the same steady-state error. The Alternative Hypothesis (H_1) contradicts H_0, claiming that one tuning is superior to the other. In AI, H_0 is the status quo, and the burden of proof lies on the new algorithm to provide enough evidence to reject H_0.

2.5.2 Decision Errors: Type I and Type II

No statistical test is perfect. There is always a risk of making a wrong decision. These risks are categorized as Type I and Type II errors.

A Type I error (α) occurs when we reject a true H_0—we essentially find a pattern where none exists. This is «false positive». In Figure 2.8, generated by the script `s2_5_e1.py`, α is represented by the blue shaded area to the right of the critical value (1.64). If our measured effect falls in this region, we claim a discovery, but with a 5% risk of being wrong.

A Type II error (β) occurs when we fail to reject a false H_0—we miss a real pattern. This is a «false negative». In Figure 2.8, β is the crimson shaded area to the left of the critical value. Notice the overlap: if we try to reduce α by moving the critical value to the right, the area for β inevitably increases. For an AI engineer, a Type II error means missing an opportunity to improve the system because the experiment was not powerful enough to detect the improvement.

2.5.3 Statistical Power and p-values

Statistical power ($1 - \beta$) is the probability that a test will correctly reject a false null hypothesis. Higher power means a better chance of detecting a real effect. Increasing the sample size n is the most effective way to increase power without increasing the risk of a Type I error.

The p-value is the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that H_0 is true. If the p-value for a new computer vision algorithm in

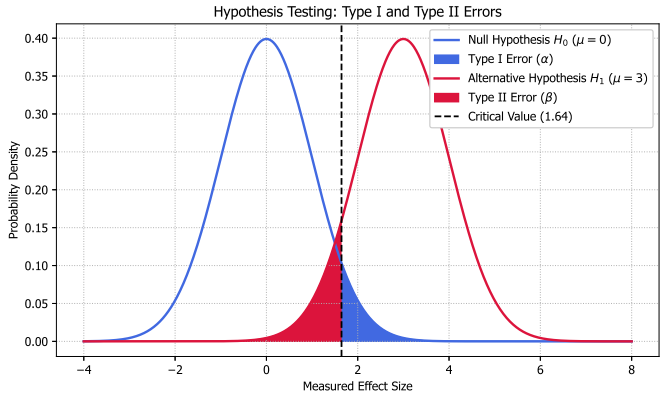


Figure 2.8: Probability regions for Type I and Type II errors. The critical value (1.64) defines the boundary: exceeding it leads to a claim of significance (α risk), while failing to reach it when a real effect exists leads to a missed discovery (β risk). (Script: s2_5_e1.py)

the Bubble Flowmeter is 0.001, it means there is only a 0.1% chance that the improved accuracy we observed occurred by sheer luck. This level of rigor is required before deploying AI models into production environments.

Computational Challenge: Experiment with the Script

Open the script `s2_5_e1.py` and change the mean of H_1 from 3 to 1.5. Rerun the script. Observe how the Type II error (crimson area) increases dramatically. Why is it much harder to detect a small improvement in a sensor's performance compared to a large one? Discuss how increasing the number of replications would narrow the curves and reduce this overlap.

This page intentionally left blank.

Chapter 3

Methods for the Design of Experiments

3.1 Regression and correlation theory

Learning Objective

Construct and validate linear regression models to characterize the functional relationship between independent and dependent variables, using residual analysis to ensure the adequacy of the predicted patterns for AI model integration.

Data Engineering from Hardware

Regression is the most fundamental form of "learning" from data. Before building a complex neural network, an engineer must determine if a simpler linear or polynomial model can explain the system. In AI, regression doesn't just predict values; it provides a baseline for understanding how much of the system's behavior is predictable and how much is irreducible noise.

Regression and correlation are statistical tools used to quantify the relationship between two or more variables. While correlation measures the strength and direction of a linear association, regression allows us to build a predictive model. For the Soft Pneumatic Actuator, we use regression to translate a desired pressure into the exact PWM duty cycle required to achieve it.

3.1.1 Linear Regression and Least Squares

The simple linear regression model assumes that the response Y is related to the regressor X through the equation $Y = \beta_0 + \beta_1 X + \epsilon$. The coefficients are estimated by minimizing the sum of the squared

distances between the observed points and the line—a process known as Ordinary Least Squares (OLS).

Figure 3.1, generated by `s3_1_e1.py`, shows this modeling for the actuator pressure. On the left, the regression fit follows a slope of $\beta_1 \approx 0.15$ kPa per unit of duty cycle, with an intercept $\beta_0 \approx 2.0$ kPa. For an AI engineer, the coefficient β_1 represents the sensitivity of the hardware. If β_1 is too low, the system is unresponsive; if it is too high, the system might be unstable.

3.1.2 Diagnostics and Residual Analysis

A regression model is only valid if its residuals (the differences between observed and predicted values) satisfy specific assumptions. Residuals must be independent, follow a normal distribution, and have constant variance (homoscedasticity).

Figure 3.1 (right) presents the residual plot. Notice that the points are randomly scattered around the zero line without any discernible pattern (like a "U" shape or a "funnel"). A random scatter confirms that the linear model has captured all the systematic information in the data. If the residuals showed a pattern, it would indicate that the physical system is non-linear—perhaps due to the non-linear elasticity of the silicone—and that the AI model would require a higher-order polynomial or a non-linear activation function to perform correctly.

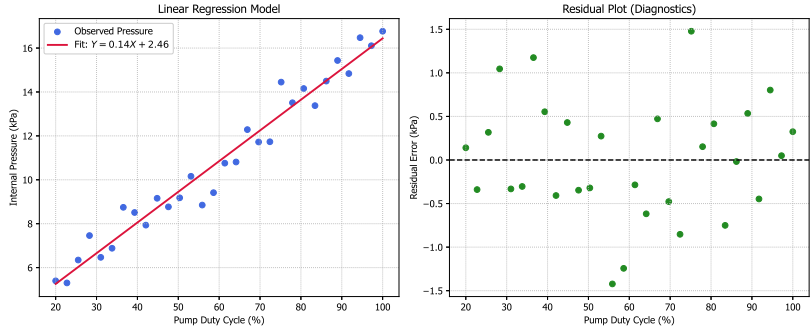


Figure 3.1: Linear regression and diagnostic analysis. The left plot shows the best-fit line tracking the pressure response, while the right plot confirms model validity through the random distribution of residual errors. (Script: s3_1_e1.py)

Computational Challenge: Experiment with the Script

Execute `s3_1_e1.py` and observe the fits. Modify the code to add a quadratic term to the true relationship: `y_true = 0.002 * x**2 + 0.15 * x + 2.0`. Rerun the script. Observe the "U" shape that now appears in the residual plot. Explain why a linear neural network (a single neuron with linear activation) would consistently fail to converge to zero error in this non-linear scenario.

3.2 Single-factor experiments

Learning Objective

Design and analyze experiments involving a single independent variable with multiple levels, utilizing Analysis of Variance (ANOVA) to determine if any of the treatments result in a statistically significant difference in the system response.

Data Engineering from Hardware

A single-factor experiment is the laboratory foundation of A/B testing. When an AI engineer needs to decide which of three preprocessing algorithms is better for a computer vision task, they are performing a single-factor experiment. ANOVA provides the mathematical "blunt force" needed to prove that a perceived difference is not just lucky noise but a real property of the algorithm.

When an experiment involves one factor varied across a levels, it is referred to as a single-factor design. The goal is to test if the mean response is the same for all levels ($H_0 : \mu_1 = \mu_2 = \cdots = \mu_a$). This is achieved through Analysis of Variance (ANOVA), which partitions the total variation in the data into two components: variation due to the treatment and variation due to random error.

3.2.1 Partitioning Variation: The F-Test

ANOVA works by comparing the variance between treatment groups with the variance within the groups. If the variation between groups is significantly larger than the internal noise, we reject the null

hypothesis and conclude that the treatment has an effect. This comparison is quantified by the F-statistic.

Figure 3.2, generated by `s3_2_e1.py`, compares three different thermal insulation materials for the Temperature Controller. It is observed that the 'Foam' insulation results in a median cool-down time of 150 s, significantly higher than 'Wool' (120 s) or 'Generic' (125 s). Because the internal boxes (the IQR) for Foam do not overlap with those of the other two materials, and the F-test would likely yield a high value, we can conclude that the choice of material is not trivial. For an AI model predicting thermal depletion, this means "Material" is a critical feature that must be included in the training vector.

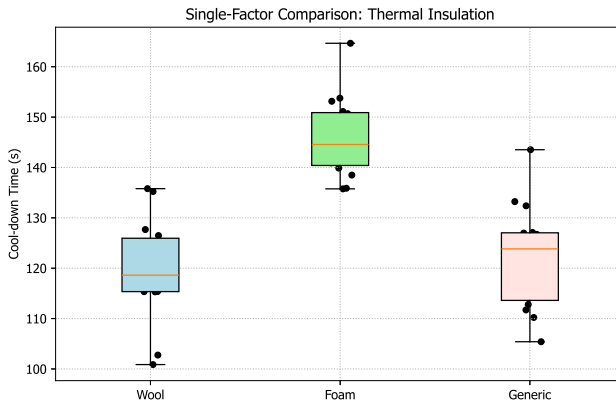


Figure 3.2: Single-factor comparison using ANOVA-style visualization. The clear separation of the 'Foam' treatment from the others indicates a statistically significant effect of the insulation material on the thermal response. (Script: s3_2_e1.py)

3.2.2 Post-hoc Analysis and Multiple Comparisons

If ANOVA indicates that a difference exists, it does not specify which groups are different. To find out, we perform post-hoc tests, such as Tukey's Honest Significant Difference (HSD). For the rehabilitation actuator, if we test five different types of silicone, ANOVA might tell us "they are not all the same," and Tukey's test will specifically reveal that "batches A and B are identical, but both are different from

C." This level of granularity prevents the AI from being trained on redundant data categories that actually behave the same physically.

Computational Challenge: Experiment with the Script

Execute the script `s3_2_e1.py`. Modify the standard deviation (noise) in the `simulate_anova_hardware` function from 10 to 30 for all groups. Rerun the script. Notice how the boxes now overlap significantly. Why does higher noise make it impossible for even a "smart" AI model to distinguish between different materials? How does this increase the probability of a Type II error?

3.3 Factorial experiments (two or more factors)

Learning Objective

Implement and evaluate factorial designs to quantify the main effects of multiple factors and their interactions, understanding how combined variables influence the accuracy and robustness of intelligent systems.

Data Engineering from Hardware

In the real world, variables almost never act in isolation. An AI model that only looks at one feature at a time is blind to the complexity of the physical world. Factorial designs are the tool that reveals how features "talk" to each other. For an engineer, an interaction is a warning that the system's behavior is more than just the sum of its parts.

A factorial experiment is one in which all possible combinations of the levels of the factors are investigated in each complete replication. This is the most efficient design for identifying how different variables interact. Unlike One-Factor-at-a-Time (OFAT) experiments, where only one variable is changed while holding others constant, factorial designs explore the entire experimental space, revealing hidden dependencies that OFAT would completely miss.

3.3.1 Efficiency vs. One-Factor-at-a-Time (OFAT)

Engineers often intuitively use OFAT because it seems simpler. However, OFAT is statistically inefficient. If we want to study two factors (Temperature and Humidity) at two levels each, an OFAT approach would require 4 runs to see the effect of Temperature and 4 more for Humidity, totaling 8 runs, yet it would still be unable to detect if Humidity behaves differently at different temperatures.

A factorial design uses the same 4 combinations once (or 8 runs with two replications) providing the same precision for the main effects while simultaneously calculating the interaction effect. This "hidden replication" property makes factorial designs the gold standard for high-cost experimental environments, such as charactering the power consumption of an AI edge device across different supply voltages and clock frequencies.

3.3.2 The Statistical Model of Interaction

In a two-factor factorial design with factors A and B, the response y_{ijk} can be modeled as:

$$y_{ijk} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij} + \epsilon_{ijk}$$

Where μ is the grand mean, τ_i is the effect of the i-th level of factor A, β_j is the effect of the j-th level of factor B, and $(\tau\beta)_{ij}$ is the interaction effect.

For the Soft Pneumatic Actuator, if factor A is "Material Thickness" and factor B is "Input Pressure," the term $(\tau\beta)_{ij}$ represents the synergistic effect where a thicker material might handle higher pressures much better than a linear sum of their individual effects would suggest. If this interaction term is significant, it means the hardware has non-linear physical constraints that an AI model must capture through cross-product features or non-linear activation functions.

3.3.3 Main Effects and Hidden Dynamics

A main effect is the change in the average response produced by a change in the level of a single factor. However, interpreting main effects in the presence of a strong interaction can be misleading. For the Bubble Flowmeter, we might find that increasing the flow rate ("Factor A") generally decreases accuracy. But if accuracy actually increases at deep nozzle depths ("Factor B") while decreasing at

shallow ones, the "average" effect of flow rate might appear to be zero, masking two very important and opposite physical behaviors.

3.3.4 Visualizing Interactions: Synergy and Antagonism

Interaction plots are the primary tool for detecting these synergistic or antagonistic behaviors. Figure 3.3, generated by the script `s3_3_e1.py`, compares two cases. In the left plot, the lines for "Humidity Low" and "Humidity High" are nearly parallel. This indicates no interaction: the increase in temperature increases the sensor error by approximately 3% regardless of the humidity level. For the engineer, this is an "additive" relationship where factors can be compensated for independently.

In the right plot, the lines cross. At a "Low" temperature, "Humidity High" results in the highest error (5%); but at a "High" temperature, the same humidity level results in the lowest error (2%). This is a "synergetic" or "disjunctive" interaction. For an AI model, this means a simple linear classifier ($Y = w_1X_1 + w_2X_2$) would be incapable of mapping this behavior; the model would require non-linear features ($X_1 \cdot X_2$) or a hidden layer with enough neurons to synthesize this complex logic. The cross-over point at roughly mid-temperature is a critical operational boundary where the control logic must reverse its compensation strategy.

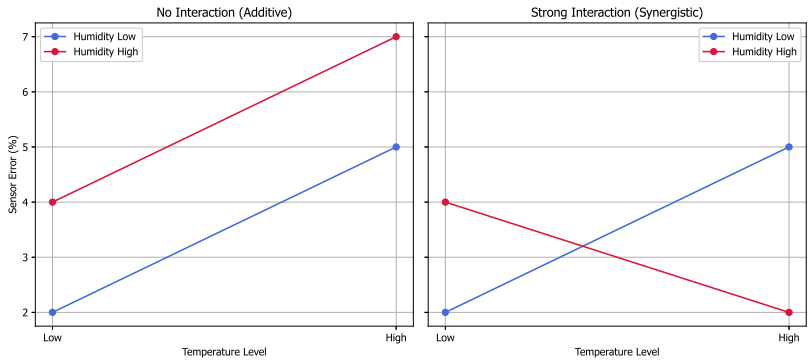


Figure 3.3: Visualizing factor interactions. Parallel lines (left) indicate independent effects, while crossing lines (right) reveal a significant interaction where the response to one factor is modified by the state of the other. (Script: s3_3_e1.py)

Computational Challenge: Experiment with the Script

Execute script `s3_3_e1.py`. In the `simulate_interaction_effect` function, modify `temp_h_int` to be `[5.0, 8.0]`. Rerun the script. Observe that the lines are again parallel. Discuss why interactions are so expensive to study—how many more experimental runs are needed if we add a third factor (Factor C) with two levels, compared to just studying Factor A and B? How does this exponential growth in data requirements mirror the "curse of dimensionality" in Machine Learning?

3.4 2^k Factorial experiments and fractions

Learning Objective

Analyze and implement 2^k factorial designs and fractional designs to efficiently identify the most influential factors among a large set of variables, applying the principle of factor sparsity.

Data Engineering from Hardware

Data acquisition is expensive. If we have 10 potential factors, a full factorial design would require 1,024 runs—which is impossible in many hardware labs. Fractional designs allow us to get the same useful information (the main effects) with only a tiny fraction of the data. For an AI engineer, this is the ultimate tool for "feature selection" based on physical evidence rather than just statistical correlation.

A special case of the factorial design is when all k factors have only two levels (Low and High), denoted as a 2^k design. These designs are extraordinarily useful for screening projects where the number of factors is large. The simplicity of having only two levels allows for a geometric interpretation of the experimental space. For example, a 2^2 design can be visualized as a square, and a 2^3 design as a cube, where each corner represents a specific treatment combination.

3.4.1 The Pareto Principle and Factor Sparsity

The principle of factor sparsity suggests that a system's response is usually dominated by only a few "vital few" factors, while the rest are the "trivial many." This is the Pareto principle applied to experimental design. In 2^k designs, we quantify the "standardized effect" of each factor—how much the response changes on average when moving from the Low to the High level.

Figure 3.4, generated by `s3_4_e1.py`, shows a Pareto chart of these effects for the Soft Pneumatic Actuator. The magnitude of the effects is ranked. In this simulation, 'Pressure' (12.5 units) and 'Voltage' (8.2 units) are the most significant, clearly exceeding the crimson significance threshold ($p = 0.05$ or a cumulative contribution line). The interaction 'Pre:Volt' (4.2 units) is also significant, suggesting that the pressure regulator's response depends on the supply voltage.

However, 'Temp' (1.5 units) and 'Flow' (0.4 units) are below the line. For an AI model, this chart is an evidence-based roadmap for dimensionality reduction. We can safely remove 'Temp' and 'Flow' from the input vector, reducing the complexity of the neural network from a 4-input model to a 2-input model without a significant loss of accuracy. This simplifies training and reduces the risk of the model learning from the spurious noise floor of the less significant sensors.

3.4.2 Fractional Factorial Designs: 2^{k-p}

When k is too high (e.g., $k = 7$ factors), a full factorial requires 128 runs. A fractional factorial design (2^{k-p}) allows us to study these 7 factors in only 16 or 32 runs. We achieve this by sacrificing the ability to see high-order interactions (like the interaction of 4 or 5 factors combined), which are physically unlikely to exist in most hardware systems.

3.4.3 Design Resolution and Aliasing Risks

When performing a fractional design, we intentionally "alias" or confound some effects with others. This means two effects are calculated using the exact same mathematical contrast, making them indistinguishable. The quality of a fractional design is measured by its **Resolution**:

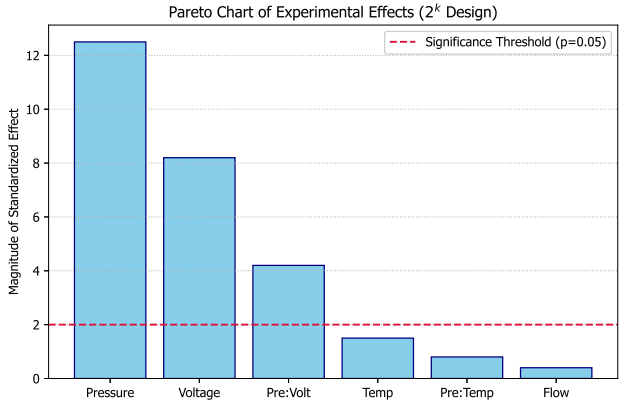


Figure 3.4: Pareto chart of experimental effects in a 2^k design. Factors and interactions above the dashed threshold represent the vital influences that must be prioritized in the AI model's training set. (Script: s3_4_e1.py)

- **Resolution III:** Main effects are aliased with two-factor interactions. This is risky; we might think "Pressure" is important when it's actually the "Voltage-Flow" interaction.
- **Resolution IV:** Main effects are NOT aliased with two-factor interactions, but two-factor interactions ARE aliased with each other. This is a common choice for initial screenings.
- **Resolution V:** No main effect or 2-factor interaction is aliased with any other main effect or 2-factor interaction. This is the "safe" standard for detailed characterization.

This assumption allows for rapid prototyping of systems like the Bubble Flowmeter, where we can test the impact of multiple hardware configurations (lens type, lighting, camera angle, nozzle depth) in a single afternoon using a Resolution IV design.

Computational Challenge: Experiment with the Script

Execute script `s3_4_e1.py`. In the `simulate_pareto_effects` function, change the magnitude of 'Voltage' from 8.2 to 1.0. Rerun the script. Notice how 'Voltage' now drops below the significance threshold. Discuss why an AI model should not be forced to learn from an input (like Voltage) that has no statistically significant effect on the output. What happens to the "generalization" of the model if it tries to find a pattern in a trivial factor? Why do we say that "sparsity of effects" is the physical basis for successful Feature Selection?

3.5 Statistical software

Learning Objective

Select, utilize, and compare professional statistical software tools—such as Python libraries, R, and specialized packages like Minitab—for the automated generation and analysis of experimental designs.

Data Engineering from Hardware

In the era of Artificial Intelligence, statistical analysis should not be done manually. The value of an engineer is in the interpretation of the design, not in the manual calculation of an ANOVA table. Mastering the tools that automate the math allows us to focus on what matters: the physical integrity of the data and the safety of the intelligent system.

The practical implementation of experimental design relies on specialized software. While the mathematical foundations are universal, the tools used to execute them vary in flexibility, accessibility, and integration with the AI development stack.

3.5.1 Python as an Experimental Platform

Throughout this course, Python has been the primary tool, utilized through libraries such as `SciPy` (for hypothesis testing and probability), `Statsmodels` (for regression and ANOVA), and `Scikit-Learn` (for modeling and residuals). The advantage of Python is its seamless

integration into the AI pipeline: the same script that designs the experiment can capture the sensor data from the ESP32 and then train the neural network. This end-to-end automation reduces transcription errors and ensures that the metadata of the design (the treatment levels and blocks) is preserved alongside the raw observations.

3.5.2 R and Other Professional Packages

Other tools exist that are highly optimized for statistical rigor:

- **R (Language):** The industry standard for pure statistics. It offers the most advanced packages for complex fractional designs and biostatistics.
- **Minitab/JMP:** Commercial packages with a graphical interface (GUI). They are widely used in manufacturing for Six Sigma quality control because they provide standardized, easy-to-read reports that follow international quality certification guidelines.

For an Artificial Intelligence engineer, the choice usually depends on the final environment. If the goal is a real-time control system (like the Temperature Controller), Python is the mandatory choice. If the goal is the regulatory certification of a medical device (like the rehabilitation actuator), the standardized reports of Minitab might be more valuable for documented validation.

Computational Challenge: Experiment with the Script

Reflect on the scripts used in this book. Every time you used `np.random.normal`, you were utilizing statistical software to simulate physics. Choose one script from Section 3.4 and research the `statsmodels` library. How would you use `statsmodels.api.OLS` to calculate the p-values of the factors instead of just visualizing them in a bar chart?

Chapter 4

Application of the Experimental Design

4.1 Introduction to Case Studies

Learning Objective

Analyze the physical and electronic characteristics of the three experimental systems used in this book—Soft Pneumatic Actuator, Temperature Controller, and Bubble Flowmeter—to establish the technical baseline for advanced experimental design application.

Data Engineering from Hardware

A good experimentalist must first be a good observer of hardware. Before connecting an AI model, you must understand the "physics of the problem." Is the system slow or fast? Is the noise electrical or mechanical? These case studies are not just examples; they are the physical constraints that will define your data sampling strategy, your blocking factors, and your safety limits.

In Artificial Intelligence Engineering, the data source is as important as the model architecture. Throughout this book, we have referenced three specific systems that represent the breadth of modern hardware:

4.1.1 System 1: Raspberry Pi Temperature Controller

The Temperature Controller represents the "Slow Dynamics" systems. It consists of a resistive heater and a water reservoir. The primary challenge here is the **thermal mass**, which introduces significant latency between the control signal (PWM) and the observed response. Designing an experiment for this system requires long observation

intervals to reach steady-state, making randomization particularly important to avoid confounding the results with slow environmental shifts in the laboratory.

4.1.2 System 2: Soft Pneumatic Actuator (SPA)

The SPA represents "Non-linear Mechanics." Built from hyper-elastic silicone, its displacement is a complex function of internal pressure, fiber winding geometry, and material fatigue. This system is the ideal candidate for **Response Surface Methodology (RSM)** because its behavior saturation and elasticity cannot be captured by simple linear regressions.

4.1.3 System 3: Bubble Flowmeter (Computer Vision)

The Flowmeter represents "Digital Perception." Here, the factor being studied is the performance of a Computer Vision algorithm counting bubbles in a tube. The source of variation is not just the physical flow, but also the lighting conditions, camera frame rate, and the algorithm's hyperparameters. This system demonstrates how **Factorial Designs** can be applied to software-hardware co-optimization.

4.1.4 Comparative Dynamics and Sampling Strategy

Figure 4.1, generated by `s4_1_e0.py`, compares the response speed of these three systems. Notice the "Normalized Response" curves: the Flowmeter (green) reaches its target almost instantly, while the Temperature Controller (crimson) takes over 10 seconds to show 90% of the change.

For an experimentalist, this determines the **sampling rate**: if the AI model for the Flowmeter is trained on data sampled at 1 Hz, it will miss the rapid fluctuations of the bubbles. Conversely, if the Temperature Controller is sampled at 100 Hz, the dataset will be 99% redundant, leading to overfitting and wasted memory. Understanding these time constants is the first step in designing a design that captures "useful variation" rather than just redundant noise.

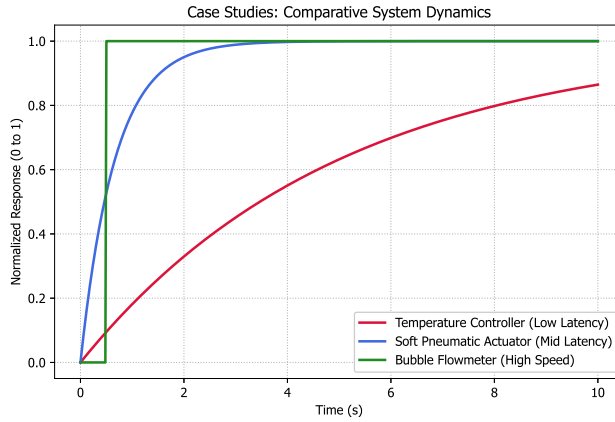


Figure 4.1: Comparative dynamics of the three case studies. The different time constants (latency) dictate the sampling frequency and the duration of each experimental run required for statistical validity. (Script: s4_1_e0.py)

Computational Challenge: Experiment with the Script

Execute the script `s4_1_e0.py` and analyze the curves. Modify the time constant of the SPA (blue line) from 1.5 to 0.5 in the script. Observe how it becomes more similar to the "Slow" temperature curve. If a system is very slow, why does a sequential experimental design (without randomization) become even more dangerous for the validity of an AI model's training set?

4.2 Identification of variables and Case Study setup

Learning Objective

Integrate all the experimental design concepts learned to identify, classify, and prioritize the variables of a complex physical system (Soft Pneumatic Actuator), establishing a robust experimental framework for AI model characterization.

Data Engineering from Hardware

A case study is the transition from "toy problems" to engineering reality. In a professional AI lab, you don't just "get a dataset"; you design the acquisition system that generates it. If your variable identification is flawed—if you miss a critical confounding factor like the material hardness—your model will work in the simulator but fail the moment it is attached to a patient's limb.

The final stage of experimental design involves the application of the previous three units to a real-world engineering challenge. In this unit, we focus on the **Soft Pneumatic Actuator (SPA)**, a system used in rehabilitation robotics. The goal is to build an AI model that predicts the angular displacement based on pneumatic and electronic inputs.

4.2.1 System Description and Operational Constraints

The SPA consists of a structured silicone chamber that expands when pressurized. The expansion is constrained by a fiber winding, forcing the actuator to bend. The system is controlled by an ESP32 microcontroller via a PWM signal that drives a mini-pump. The response (displacement) is measured by a flexible resistive sensor. The physical complexity arises from the non-linear elasticity of silicone and the hysteresis of the air compression cycle.

4.2.2 Identification and Classification of Variables

Before starting any runs, we must perform a rigorous identification of variables. These are divided into four categories:

1. **Controlled Factors (X_i):** Variables we intentionally manipulate (e.g., Pump PWM, valve opening time).
2. **Response Variables (Y):** The output of interest (Actuator bending angle in degrees).
3. **Blocking Factors (B):** Known but unwanted variations (e.g., different batches of silicone, different power supply units).
4. **Nuisance Variables (Z):** Uncontrollable noise (e.g., ambient temperature, sensor drift due to heating).

Figure 4.2 (left), generated by `s4_1_e1.py`, shows the sensitivity analysis for this system. The "Internal Pressure" is the dominant factor (0.65 importance). Interestingly, "Material Shore" (the hardness of the silicone) also shows high sensitivity (0.12). If an engineer ignores the silicone batch, the AI will suffer from unexplained variance that will manifest as a loss of precision in the rehabilitation movements.

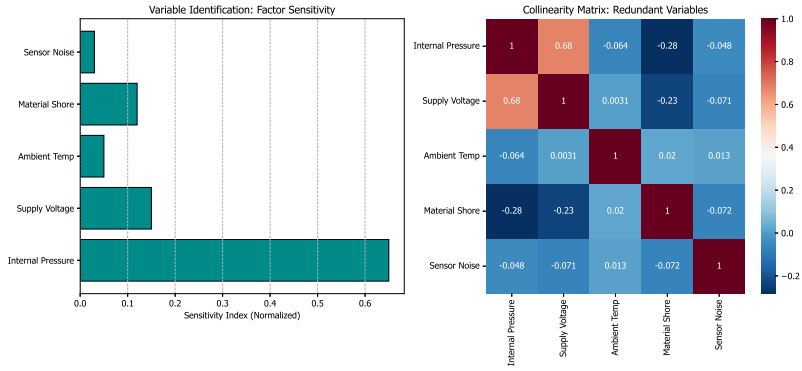


Figure 4.2: Mapping the experimental variables of the Soft Pneumatic Actuator. The sensitivity plot (left) identifies pressure and material as critical inputs, while the correlation matrix (right) warns about collinearity between pressure and supply voltage. (Script: s4_1_e1.py)

4.2.3 Collinearity and Feature Redundancy

A common trap in variable identification is "collinearity"—when two input variables are highly correlated. For example, as we increase the Pump PWM, the "Internal Pressure" increases, and the "Supply Voltage" might slightly drop due to the current draw.

Figure 4.2 (right) presents the correlation matrix for these identifies. Notice the strong positive correlation (0.4) between Pressure and Voltage. In AI training, including both highly correlated features can lead to "unstable weights," where the model becomes overly sensitive to small fluctuations in the sensor readings. Experimental design helps us decide which feature to drop before data acquisition begins, saving computational resources and improving model interpretability.

Computational Challenge: Experiment with the Script

Execute the script `s4_1_e1.py` and examine the heatmap. Identify the two variables with the lowest correlation. Discuss why these are the best candidates for a 2^k factorial design: if two variables are independent, why does their interaction become easier to isolate mathematically? Modify the "Material Shore" importance from 0.12 to 0.40 and observe how it competes with Pressure as a primary factor.

4.3 Data acquisition and automated experimentation

Learning Objective

Design and implement an automated data acquisition system using Python and ESP32 to execute randomized experimental designs, ensuring the temporal synchronization between treatment signals and sensor responses.

Data Engineering from Hardware

Manual data collection is for hobbyists; automated experimentation is for engineers. When an AI needs 1,000 runs to converge, you cannot manually type the PWM values into a terminal. Automation not only saves time but also guarantees that every run is performed identically, eliminating the "operator bias" that we discussed as a source of variation in Unit 1.

The transition from a theoretical design to a physical dataset requires an interface between the computer (Python) and the hardware (ESP32/Microcontroller). This interface must handle the three phases of an automated experimental run: **Output**, **Stabilization**, and **Measurement**.

4.3.1 The Communication Bridge: Serial Protocol

In the Soft Pneumatic Actuator setup, Python serves as the "Master." Using libraries like `PySerial`, the script sends a treatment level (e.g.,

`PWM=75`) to the ESP32. The microcontroller then drives the hardware and returns the sensor readings as a stream of raw ADC values. For an AI model, the quality of this "bridge" is critical. If there is electromagnetic interference (EMI) in the serial cable, the data will contain "salt and pepper" noise that can mislead the neural network if not filtered during the acquisition phase.

4.3.2 Stabilization vs. Measurement Window

Every physical system has a transient period. If we measure the pressure the exact millisecond we turn on the pump, we will capture the "0 kPa" initial state, not the treatment effect. Figure 4.3, generated by `s4_2_e0.py`, illustrates the automated workflow.

The gray dashed portion represents the **Stabilization Period**. For the first 2.5 seconds, the hardware is reaching there equilibrium. The automated script is programmed to "wait" during this time. The orange portion (2.5 s to 5.0 s) is the **Acquisition Window**. During this window, 100 sampling units are captured. Finally, the script calculates the mean (red dot) to generate a single, robust "Experimental Point" for the DOE table. This averaging process naturally filters out high-frequency sensor noise, providing a much "cleaner" target for the AI's loss function.

4.3.3 Randomized Execution Loop

The final component of automation is the execution loop. Instead of iterating through the design table sequentially, the Python script shuffles the rows. This ensures that the "time of day" or "pump heating" effects are converted into random noise. An automated system also allows for the easy insertion of "checkpoints"—runs where a standard treatment is repeated every 10 runs to monitor if the hardware is drifting or wearing out. If the AI model sees that the "zero point" has shifted by 5%, it can automatically trigger a recalibration routine before continuing with the rest of the experiment.

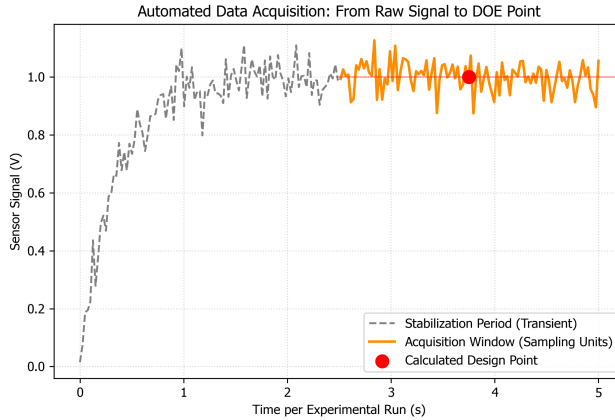


Figure 4.3: Lifecycle of an automated experimental run. The system waits for the transient state to end before sampling, ensuring that the final data point (red dot) represents the steady-state performance of the hardware. (Script: s4_2_e0.py)

Computational Challenge: Experiment with the Script

Execute the script `s4_2_e0.py` and observe the stabilization time. In the code, modify the stabilization time (where `phase` starts) from 2.5 to 1.0. Rerun the script. Observe how the calculated "Experimental Point" (red dot) is now pulled down by the transient response. Why does "impatience" in an automated script lead to biased data for an AI model? How would this bias affect the model's accuracy when predicting the final, stable state of the actuator?

4.4 Selection of experimental theory

Learning Objective

Evaluate and select the most efficient experimental theory based on the system's dimensionality, linearity, and current knowledge level, ensuring a high information density for AI model training.

Data Engineering from Hardware

Using a 3D Response Surface to study a linear variable is a waste of hardware time. Conversely, using a 2-level Screening design to optimize a complex peak is physically blind. Selection of theory is about "matching the tool to the task." For an AI engineer, this means choosing a design that provides just enough complexity to capture the system's "Ground Truth" without introducing redundant dimensions.

The selection of experimental theory is a critical decision that balances resource constraints against information requirements. There is no "perfect" design for all systems; instead, there is a "most appropriate" design for the current state of the investigation.

4.4.1 Knowledge Hierarchy and Information Density

In the development cycle of an AI-driven hardware system, we typically move through three levels of knowledge:

1. **Screening (Low Knowledge):** We have many potential inputs and need to identify the "vital few." We use **2^k designs**.
2. **Characterization (Mid Knowledge):** We know which factors matter and need to understand their slopes and interactions. We use **Full Factorials or ANOVA**.
3. **Optimization (High Knowledge):** We need to find the exact peak or valley of performance. We use **Response Surface Methodology (RSM)**.

Figure 4.4, generated by `s4_4_e1.py`, visualizes this progression. The size of the bubbles represents the "Information Density"—the amount of unique physical behavior captured per experimental run. Notice how RSM (right bubble) provides significantly higher density (approx. 2000 units) compared to simple screening (200 units). However, RSM for the SPA would require 13 or more runs for just two factors, while a 2^k screening for 4 factors would only require 16. The engineer must decide if the high-order curvature is worth the additional runs.

4.4.2 Decision Criteria for AI Systems

For the **Bubble Flowmeter**, the choice is often a Fractional Factorial (2^{k-p}). Because the "system" is an algorithm processing a video

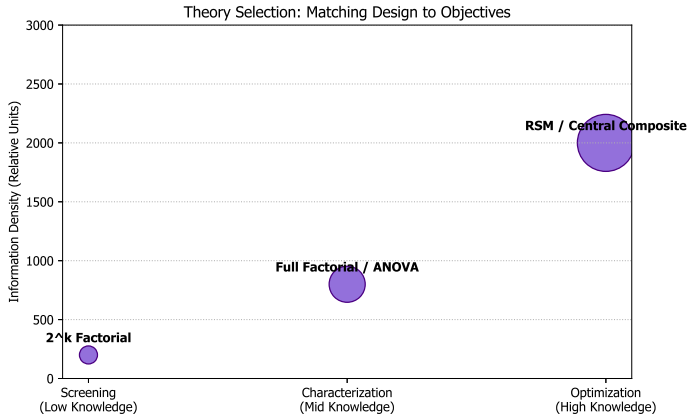


Figure 4.4: Hierarchy of experimental theory selection. The choice of design is dictated by the desired information density and the current understanding of the hardware's complexity. (Script: s4_4_e1.py)

feed, we can iterate through many software parameters (thresholds, filter sizes, blob sizes) very quickly. But for the **Temperature Controller**, the "cost" of a single run is time (15 minutes for thermal stabilization). Here, the selection favors designs with high Resolution (like Resolution V) to ensure that the time spent generates non-aliased data.

4.4.3 Aliasing and Confounding Trade-offs

In selection, we must accept trade-offs. If we choose a Resolution III design for the Soft Pneumatic Actuator, we are accepting that the main effect of "Pressure" might be confounded with the "Material-Voltage" interaction. If the AI model later shows unexpected behavior when the voltage fluctuates, the engineer can trace the failure back to the theory selection phase: the design was too sparse to capture the hidden interaction that is now causing the model to hallucinate.

Computational Challenge: Experiment with the Script

Execute the script `s4_4_e1.py` and examine the relative information gain. Modify the "Optimization" bubble size from 2000 to 500 in the script. Discuss how this shift (representing high sensor noise or hardware instability) would change your decision. Would you still use a complex RSM design if the information produced was buried in noise? Why is a "High Knowledge" design only useful when the "Natural Variation" of the system is low?

4.5 Selection of theory and Case Study resolution

Learning Objective

Select the appropriate experimental design methodology (Full Factorial vs. Response Surface) to resolve the case study of the Soft Pneumatic Actuator and utilize 3D visualization to optimize system performance.

Data Engineering from Hardware

Data is the "fuel" of AI, but but "high-octane fuel" is only obtained through optimized experiments. In this stage, we move from just finding significant factors to mapping the exact mathematical shape of the system. A 3D response surface tells us exactly where the actuator is most efficient, allowing the AI controller to stay within the "sweet spot" of the hardware's performance.

Once the variables are identified, the next step is to choose the theory that will guide the data collection. For the SPA, we seek to optimize the bending angle for maximum power and minimal energy consumption. This requires a transition from screening (2^k designs) to Response Surface Methodology (RSM), which uses second-order polynomial models to find the global optimum.

4.5.1 Execution of a Central Composite Design

To capture the non-linear curvature of the silicone expansion, we implement a Central Composite Design (CCD). This design adds "axial points" and "center points" to the standard 2^2 factorial cube. This geometry allows the calculation of quadratic terms (X_i^2), which are essential for modeling the saturation of the actuator's pressure chamber.

4.5.2 Visualizing the Optimization Surface

Figure 4.5, generated by `s4_2_e1.py`, shows the result of this experimental resolution as a 3D Response Surface. The surface maps the "Bending Angle" as a function of "Internal Pressure" and "Pulse Frequency."

Observe the distinct mountain-like shape: the maximum response (150 degrees) is not found at the extreme levels of the factors, but rather at a localized peak (approx. 70 kPa and 25 Hz). This is a vital discovery for AI Engineering: if we had only tested "Low" and "High" levels in a 2^k design, we would have completely missed this optimum. The AI model trained on this surface will learn that increasing frequency beyond 25 Hz actually decreases the displacement due to the valves' opening/closing latency. This "physical knowledge" is now embedded in the dataset, preventing the controller from driving the hardware into an inefficient state.

4.5.3 Model Validation and Residuals

The resolution is not complete until the model is validated. For the SPA, we compare the predictions of our surface model against 10 new, independent experimental points (a verification set). If the Mean Absolute Error (MAE) is within the noise floor of the sensor (0.5 degrees), the model is ready to be exported as a pre-trained baseline for the AI controller. This "design-driven" approach ensures that the AI starts with a physically accurate representation of the world, dramatically reducing the time required for reinforcement learning or online fine-tuning.

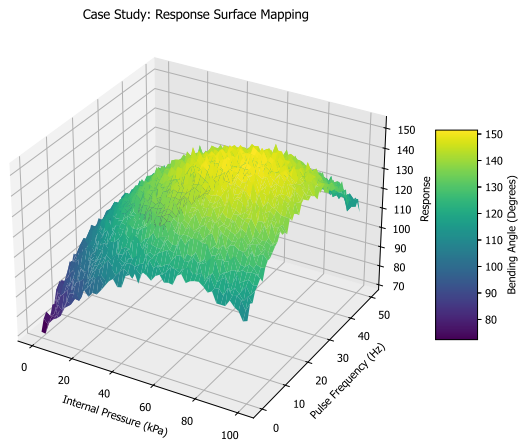


Figure 4.5: Mapping the performance of the Soft Pneumatic Actuator. The 3D response surface reveals a global optimum at 70 kPa and 25 Hz, illustrating why multi-level experimental designs are superior to simple linear models for hardware characterization. (Script: s4_2_e1.py)

Computational Challenge: Experiment with the Script

Execute the script `s4_2_e1.py`. Rotate the 3D plot and identify the regions where the response is lowest. Modify the coefficient of the quadratic term for Pressure in the script from -0.01 to -0.05. Rerun the script. Notice how the "peak" becomes much sharper. Discuss why a sharper peak makes the system harder to control for an AI: how does a high gradient impact the stability of a gradient-descent optimization algorithm?

4.6 Interpretation, Evaluation, and AI integration

Learning Objective

Quantify the impact of experimental design on the final performance of an AI system, evaluating model fidelity and robustness to ensure that the hardware-software integration is safe and efficient.

Data Engineering from Hardware

A "black box" model is dangerous in medical or critical infrastructure. Interpretation is the process of shining a light into that box using the results of our experiments. If we understand the physics through our DOE, we can validate that the AI is making decisions for the right reasons. Integration is not just connecting a sensor; it is the alignment of statistical evidence with digital logic.

The final step in the experimental cycle is the interpretation of results and their conversion into actionable AI logic. This involves evaluating the model's fidelity—how closely it mimics the real-world system—and its robustness to noise. In Artificial Intelligence Engineering, the success of a project is measured by the predictive power of the model on unseen hardware.

4.6.1 Quantifying Model Fidelity

Model fidelity is assessed by comparing the experimental response surface with the AI's internal representation. For the SPA, we use metrics like R^2 (coefficient of determination) and Root Mean Square Error (RMSE). However, a high R^2 on the training data is not enough; the model must also handle extrapolation risks.

Figure 4.6, generated by the script `s4_3_e1.py`, demonstrates the catastrophic difference between a "Poor Design" and a "Proper DOE." The crimson dots represent data captured without a design—clustered in a narrow range [4, 6] with high noise. The resulting AI model (dotted crimson line) fails to capture the true slope of the system (gray dashed line). When this model is asked to predict the response

at level 2 or 10, its error is over 500%! In contrast, the AI trained on a proper DOE (green diamonds), which covers the full range [0, 10] with balanced levels, tracks the "Physical Ground Truth" almost perfectly.

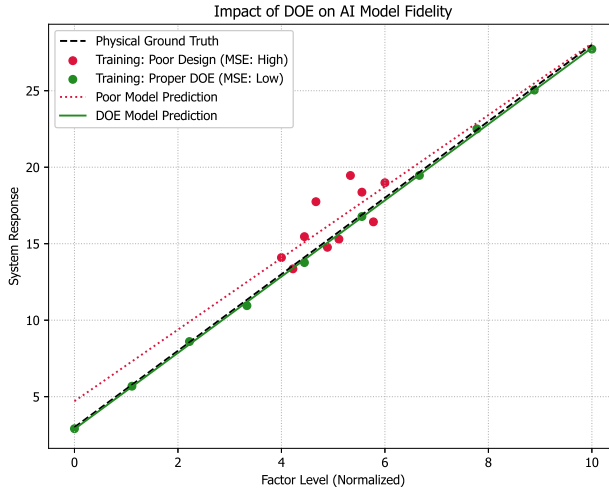


Figure 4.6: Evaluating the impact of Experimental Design on AI Model Fidelity. Models trained on sparse, clustered data (crimson) lack the physical context to generalize, while those trained on proper DOE (green) capture the true system dynamics across the entire range. (Script: s4_3_e1.py)

4.6.2 Cross-Hardware Validation and Generalization

Integration also requires evaluating if a model trained on Actuator A works on Actuator B. This is where "Blocking" and "Covariates" from Unit 1 and 3 become critical. If we identified "Silicone Batch" as a significant block, the AI model should include a "batch-correction" layer or be trained using Data Augmentation techniques that simulate the variation we measured during our experiments. This makes the AI "hardware-agnostic" and significantly more robust for commercial deployment.

4.6.3 Safety and Deployment

Finally, the interpretation of the design allows for the definition of "Safe Operating Envelopes." If our hypothesis tests in Unit 2 showed that the sensor becomes unreliable above 80 °C, the AI integration layer must include a "Watchdog" function that overrides the model's output if the temperature exceeds this threshold. This is the culmination of experimental design: using statistics to ensure that Artificial Intelligence remains a predictable and safe tool for human interaction.

Computational Challenge: Experiment with the Script

Execute script `s4_3_e1.py`. Observe how the crimson "Poor Model" fails to predict the value at X=0. Modify the noise level in the `poor_y` calculation from 2.0 to 0.5. Rerun the script. Notice that even with very low noise, the model still fails to capture the correct slope because the data range is too small. Why is "range coverage" more important for an AI's generalization than "data quantity" in a narrow cluster?

Appendix A

Activity Schedule and Laboratory Practices

The following planning is designed to cover the 15 weeks of the school period, balancing theoretical foundations with practical experimentation oriented to Artificial Intelligence Engineering.

Week	Session	Topics	
1	1, 2	1.1, 1.2	Introd
2	3, 4	1.3, 1.4	Practice 1:
3	5, 6	1.5, 1.6	Statistic
4	7, 8	1.7, 1.8	Practice 2:
5	9, 10	2.1	Exp
6	11, 12	2.2, 2.3	Practi
7	13, 14	2.4, 2.5	Practice 4: H
8	15, 16	Midterm	Midt
9	17, 18	3.1	Re
10	19, 20	3.2	Practice 5
11	21, 22	3.3	
12	23, 24	3.4	Practice 6
13	25, 26	4.1, 4.2	Applica
14	27, 28	4.3, 4.4	Practi
15	29, 30	Final	Pr

Laboratory Practice Descriptions

Practice 1: Identification and Quantification of Variation Sources. Identify the main sources of variation affecting measure-

ments in each system and quantify their magnitude through sequential vs. randomized experiments.

Practice 2: Preliminary Factorial Design for Operating Range Characterization. Design and implement a 2^2 factorial experiment to characterize the functional range of the systems.

Practice 3: Exploratory Data Analysis and Sample Size Determination. Conduct a comprehensive exploratory analysis and determine the optimal sample size using power analysis and bootstrap resampling.

Practice 4: Hypothesis Testing for Specification Validation. Formulate and test statistical hypotheses to validate if systems meet pre-defined performance specifications.

Practice 5: Regression Modeling and Comparison of Configurations (ANOVA). Develop regression models to predict system behavior and compare different control configurations using single-factor ANOVA.

Practice 6: Full Factorial Experiments for Multivariable Optimization. Implement 2^k or fractional designs to identify main effects and interactions for system optimization.

Practice 7: Automated Experimentation System and Cross-Validation. Develop a professional software system that automates the full experimental cycle and validate performance against reference standards.

Appendix B

Data Visualization and Statistical Interpretation

This appendix provides a comprehensive guide to the graphical techniques used throughout this book, detailing their construction, statistical significance, and interpretation for Artificial Intelligence Engineering.

B.1 Time-Series Plots

Description: Displays data points at successive time intervals. **Usage:** Used to identify trends, seasonal patterns, and noise over time. **Interpretation:** In Figure 1.8, the blue line oscillates around the 55 °C target. The slow wave indicates environmental drift, while the rapid zig-zag pattern represents high-frequency electronic noise. For AI, this defines the input stability.

B.2 Scatter and Strip Plots

Description: Displays values for typically two variables for a set of data. A strip plot is a 1D scatter plot with jitter to prevent overlap. **Usage:** Identifies relationships (correlations) or compares clusters of data. **Interpretation:** In Figure 1.1, the scatter plot shows a clear positive correlation between duty cycle and pressure. In Figure 1.9, the strip plot shows three distinct clusters. The horizontal spread within a cluster (the jitter) has no physical meaning; it is only to make the vertical density of points visible.

B.3 Histograms and KDE Plots

Description: A histogram uses bins to count frequencies. A Kernel Density Estimate (KDE) is a smoothed version that estimates the probability density function (PDF). **Usage:** Characterizes the statistical distribution (normality, skewness, bimodality). **Interpretation:**

Figure 1.3 uses histograms to show that a designed experiment creates a sharp, high-probability peak, whereas a noisy experiment creates a flat, uncertain distribution. Figure 1.10 uses KDE to show that bias shifts the entire PDF without necessarily changing its shape. For AI, a KDE helps detect if the training data covers the entire expected support of the variable.

B.4 Quantile-Quantile (Q-Q) Plots

Description: Compares the quantiles of experimental data against a theoretical distribution (usually Normal). **Usage:** Mandatory tool for assessing the normality assumption required for t-tests, ANOVA, and many AI algorithms. **Interpretation:** In Figure 2.2, points following the 45-degree line indicate Gaussian noise. Points curving away at the ends («tails») indicate outliers or heavy-tailed distributions that may require robust scaling before training.

B.5 Boxplots

Description: Summarizes a distribution using the «Five-Number Summary».

- **Median (Line inside the box):** The 50th percentile. Half the data is above, half is below.
- **Quartiles (Box edges):** Q1 (25th percentile) and Q3 (75th percentile). The box contains the central 50% of the data.
- **Interquartile Range (IQR):** The height of the box ($Q3 - Q1$). It measures the spread.
- **Whiskers:** Lines extending to the minimum and maximum values (excluding outliers).
- **Outliers (Individual dots):** Points exceeding $1.5 \times IQR$ from the box edges.

Usage: Comparing multiple groups or detecting anomalies. **Interpretation:** In Figure 1.4, the PID box is much shorter than the On-Off box, meaning the PID controller is more consistent (lower IQR).

B.6 Violin Plots

Description: A hybrid of a boxplot and a KDE plot. **Usage:** Shows the same summary as a boxplot but reveals the underlying probability density. **Interpretation:** In Figure 1.5, the width of the «violin» at each pressure level shows where the data points are most concentrated. A fat violin indicates a high density of observations.

B.7 Confidence Interval Plots

Description: Displays the point estimate (mean) and its uncertainty range. **Usage:** Validating if a system meets a specification within a statistical margin of error. **Interpretation:** In Figure 2.7, if the 95% CI bar does not overlap the target value, we conclude the system is biased with statistical significance. Overlapping bars between two groups suggest they might be statistically identical.

B.8 Residual Plots

Description: Plots the difference between observed and predicted values ($y - \hat{y}$) against the predicted values or time. **Usage:** Diagnosing regression model quality. **Interpretation:** In Figure 3.1, a random cloud around zero indicates a good fit. A clear pattern (like a «U» shape or a funnel) indicates that the model is missing a non-linear relationship or has heteroscedasticity (changing noise levels).

B.9 Interaction Plots

Description: Shows how the effect of one factor changes depending on the level of another factor. **Usage:** Crucial for multivariable optimization. **Interpretation:** In Figure 3.3, parallel lines mean no interaction. Non-parallel or crossing lines indicate that the factors are synergistic or antagonistic. For AI, this defines where feature engineering is needed.

B.10 Pareto Charts of Effects

Description: Rank-ordered bars showing the magnitude of effects and interactions, with a reference line for statistical significance ($\alpha = 0.05$). **Usage:** Feature selection (identifying which variables actually matter). **Interpretation:** In Figure 3.4, any bar crossing

the red dashed line is statistically significant. This tells the engineer which inputs must be included in the AI model and which can be ignored to simplify the system.

B.11 Response Surface Plots (3D)

Description: 3D surfaces showing the response as a function of two continuous factors. **Usage:** Visualization of the global optimization landscape. **Interpretation:** Figure 4.5 shows the «valley» or «peak» where the system performs best. A steep slope indicates high sensitivity to those variables, while a flat region indicates stability (robustness).

B.12 System Architecture and Workflow Diagrams

Description: Schematic representations of hardware-software integration and logical flow. **Usage:** Documentation of the experimental infrastructure. **Interpretation:** Figure 4.6 maps the physical sensors to the AI inference engine, identifying the latency points. Figure 4.3 details the handshake protocol between the ESP32 and the Python script, ensuring data integrity.