**Experimental Design definition**

Experimental design is the process in which we carry out research in an objective and controlled fashion. The purpose of this is to ensure we can make specific conclusions in reference to a hypothesis we have.

1. 1 https://www.sciencedirect.com/topics/earth-and-planetary-sciences/experimental-design

**Forming robust statements**

Because we use objective tools, we need to use quantified language. Instead of using words like 'probably', 'likely', and 'small' when noting our conclusions, we should use precise and quantified language. This often takes the form of noting the percentage risk on a Type I error in the conclusion. Recall that Type I errors occur when we incorrectly reject the null hypothesis when it is actually true. In this course, you'll learn to design experiments and conduct statistical analyses such that you begin making precise statements about observed results and take informed actions as a result.

**Why experimental design?**

Experimental design is useful in many fields. Naturally, it is used in academia such as in medical research. It is also useful in many corporate contexts such as marketing and product analytics, which conduct lots of A/B tests. It is also used in agriculture and increasingly in government policy through the use of behavioral psychology experiments.

**Some terminology...**

Before we begin our first topic, let's define some important terminology. Subjects are what we are experimenting on. It could be people, employees, or users on a website.

A treatment is some change given to one group of subjects.

We could call that group the treatment group.

The control group is not given any change. This could be a placebo group, for example.

**Assigning subjects to groups**

An important concept in experimental design is how to assign subjects to test groups. There are two ways we could do this. We could just split the dataset non-randomly into chunks and assign each chunk to a group. Or we could use random assignment to sample into our desired groups. Let's look at each option using a DataFrame of 200 subjects' heights where we want to split into two groups of 100 each.

**Non-random assignment**

Let's try non-random assignment first. We can use .iloc[] to slice the first 100 rows from heights and assign to group1 and the next 100 rows into group2. We can use pandas' describe method to check descriptive statistics of our groups. Concatenating the two results with pd.concat() and axis=1 will allow for easier comparison. These groups appear very different! Looking at the mean row, we can see there's a 9cm difference. Because of the differences in these groups, it will be harder to confidently determine if any changes are due to the treatment intervention.

**Random assignment**

Let's now try random assignment. We can use pandas' sample method to create a sample of size n, or use the frac argument and specify a proportion of the dataset, between 0 and 1, to sample. We want two equally-sized groups, so we specify frac=0.5. Using n=100 would also work here. We also set the replace argument to False, so samples aren't selected twice. The random\_state argument allows the splits to be consistently reproduced. group2 can be made by dropping the ids in group1 from the overall DataFrame. Using the same comparison method we see much closer means.

**Assignment summary**

This demonstrates the importance of randomly assigning subjects to groups. It means we can attribute observed changes to treatment interventions rather than natural differences between the group. We can use pandas' sample method to select randomly from a DataFrame, and then use pandas' describe method to check differences in group assignment.

**Experimental data setup**

We've seen that randomization is often the best technique for setting up experimental data, but it isn't always.

**The problem with randomization**

There are several scenarios where pure randomization can lead to undesirable outcomes. Firstly, when it results in uneven numbers of subjects in different groups, often seen more in smaller experiment sizes.

**The problem with randomization**

Covariates are variables that potentially affect experiment results but aren't the primary focus. If covariates are highly variable or not equally distributed among groups, randomization might not produce balanced groups. This imbalance can lead to biased results. Overall these make it harder to see an effect from a treatment, as these issues may be driving an observed change.

**Block randomization**

A solution to our uneven problem is block randomization. This involves splitting into a block of size n first, then randomly splitting. This is what it looks like. Subjects are split into two groups, then randomly assigned to be Treatment (orange) or control (white). This fixes the uneven issue, and the smaller blocks give us more control over the allocation.

**Our dataset**

Let's give block randomization a go on a dataset of 1000 members from an e-commerce site that contains variables for their average basket size in dollars, the average time spent on the website each day, and whether they are a power user. Power users spend an average of 40+ minutes on the website each day. There are 100 power users in these 1000 subjects.

**Block randomization in Python**

We can use pandas' sample method to randomly assign subjects into two blocks. A block column has also been added to both DataFrames for convenience. This produces even block sizes, fixing the uneven issue, but let's check for covariates.

**Visualizing splits**

A nice way of checking for potential covariate issues is with visualizations. We can use seaborn's displot function to produce a kde (or kernal density) plot to visualize the distribution of the basket size, split by whether the user is a power user. There is quite a difference in the group distributions. It seems like the power\_user variable could have an effect on basket size. When an effect could be because of a variable rather than the treatment, this is often called confounding. The covariate issue can be solved with stratified randomization.

**Stratified randomization**

Stratified randomization involves splitting based on a potentially confounding variable first, followed by randomization. This is what it may look like. Firstly, we split into two blocks (sometimes called strata) of power users, in green, and non-power users, in yellow. Then, inside the groups, randomly allocating to treatment or control. This fixes the uneven covariate issue, and can even be done for multiple covariates, but managing more strata does increase complexity.

**Our first strata**

Let's stratify our power users. We separate them out first and label the block. We then sample half the power users to be in Treatment. The T\_C column notes this status. We then place the remaining into control by dropping the subjects in the treatment group.

**The second strata**

For our other strata, we separate out non-power users first and label the block differently. The rest of the code is the same as before. We allocate half to treatment and control using the same column headers.

**Confirming stratification**

Let's bring our work together by firstly concatenating the strata and groups. We can confirm our work using groupby and chaining the .size() method. This will show the number of power users in each block by their treatment or control status. We can see two blocks: one with all 100 power users and another with the other 900 users, split evenly into treatment and control groups.

**Normal data**

Let's review the concept of normal data and how it relates to experimental analysis.

**The normal distribution**

Normal data is drawn from a normal distribution, which has the familiar bell curve shape. The normal distribution is intrinsically linked to z-scores, which recall, is a standardized measure of how many standard deviations a value is from the population mean. The most common normal distribution used for z-scores has a mean of zero and a standard deviation of one. This answers questions such as 'How many standard deviations is this point from the mean?' and 'What is the probability of obtaining this score?'.

**Normal data and statistical tests**

Normal data is an underlying assumption for many statistical tests, called parametric tests. There are also nonparametric tests that don't assume normal data.

**Normal, Z, and alpha**

In hypothesis testing, alpha, or the significance level, is often closely linked to the normal distribution. For normal data, we can visually see the risk of error for a given significance level and compare that result to the p-value, which is related to the z-score. An alpha of 0.05 on a standard two-tailed test represents a small region in the tails. It means there is a 5% risk of rejecting the null hypothesis when it is actually true - a so-called Type I error.

**Visualizing normal data**

We can visually check data for normality using a kde (or kernel density) plot, available via Seaborn's displot() function. On this salaries dataset, the data appears approximately normal.

**QQ plots**

A more statistically robust visual tool is a quantile-quantile, or QQ, plot. It plots the quantiles or sections of two distributions against each other. The qqplot function from statsmodels plots our data. Setting the dist argument to the normal distribution from scipy.stats compares our data against a standard normal distribution. If the distributions are similar, the dots in the QQ plot hug the line tightly. Our data again seems quite normal. Here is another example. The dots bow out at the ends, which means that the data is not very normal.

**Tests for normality**

There are also various numerical hypothesis tests for normality. The Shapiro-Wilk test is known to be good for small datasets. The D'Agostino K-squared test uses kurtosis and skewness to determine normality. These terms relate to the symmetry and size of a distribution's tails, respectively. Anderson-Darling is another common test which returns a list of values, rather than just one so we can see normality at different levels of alpha. Each of these tests has a null hypothesis that the provided dataset is drawn from a normal distribution.

**A Shapiro-Wilk test**

Let's run one of these tests, the Shapiro-Wilk test. We import it from scipy.stats, and set our alpha at 0.05. The function takes a series of values and returns a test statistic and p-value. The p-value is greater than alpha, so we have evidence our data that looked quite normal is normal. We fail to reject the null hypothesis and have evidence that the data sample is normal at the alpha level of 0.05.

**An Anderson-Darling test**

To implement an Anderson-Darling test, we provide data and set the dist argument to norm to test for normality. The result object contains a test statistic and a range of critical values and significance levels. To interpret, we check the test statistic against each critical value. If the test statistic is higher than the critical value, the null hypothesis is rejected at that particular significance level, and the data is not normal. 0.2748 is less than all the critical values, so we fail to reject the null hypothesis and suspect that the data is normal.