Hypothesis testing in Python

Let's take a coin-flipping example.

Let’s say that we flipped 100 coins and observed 70 heads. We would like to use these data to test the hypothesis that the true probability is 0.5. First let’s generate our data, simulating 100,000 sets of 100 flips. We use such a large number because it turns out that it’s very rare to get 70 heads, so we need many attempts in order to get a reliable estimate of these probabilties. This will take a couple of minutes to complete.

In [1]:
```
import numpy as np
import pandas as pd

num_runs = 10000


def toss_coins_and_count_heads(num_coins=100, p_heads=0.5):
    """
    flip a coin num_coins times and return number of heads
    """

    flips = np.random.rand(num_coins) > (1 - p_heads)
    return(np.sum(flips))


flip_results_df = pd.DataFrame({'n_heads': np.zeros(num_runs)})

for run in range(num_runs):
    flip_results_df.loc[run, 'n_heads'] = toss_coins_and_count_heads()
```

Now we can compute the proportion of samples from the distribution observed when the true proportion of heads is 0.5.

In [6]:
```
import scipy.stats

pvalue = 100 - scipy.stats.percentileofscore(flip_results_df, 70)
print(pvalue)
```

```
0.0
```

For comparison, we can also compute the p-value for 70 or more heads based on a null hypothesis of Pheads=0.5, using the binomial distribution.

compute the probability of 69 or fewer heads, when P(heads)=0.5

In [3]:
```
p_lt_70 = scipy.stats.binom.cdf(k=69, n=100, p=0.5)
p_lt_70
```

Out[3]:
```
0.999960749301772
```

The probability of 70 or more heads is simply the complement of p_lt_70

In [4]:
```
p_ge_70 = 1 - p_lt_70
p_ge_70
```

Out[4]:
```
3.925069822796612e-05
```

Simulating p-values

In this exercise we will perform hypothesis testing many times in order to test whether the p-values provided by our statistical test are valid. We will sample data from a normal distribution with a mean of zero, and for each sample perform a t-test to determine whether the mean is different from zero. We will then count how often we reject the null hypothesis; since we know that the true mean is zero, these are by definition Type I errors.

In [5]:
```

num_runs = 5000


# create a function that will take a sample
# and perform a one-sample t-test
def sample_ttest(sampSize=32):
    """
    perform a ttest on random data of n=sampSize
    """

    ttresult = scipy.stats.ttest_1samp(np.random.normal(loc=0.0, scale=1.0,
    return(ttresult.pvalue)


# create input data frame for the function
sim_results_df = pd.DataFrame({'p_value': np.zeros(num_runs)})

# perform simulations
for run in range(num_runs):
    sim_results_df.loc[run, 'p_value'] = sample_ttest()

p_error = sim_results_df['p_value'] < 0.05
p_error = p_error.mean(axis=0)
p_error
```

Out[5]:
```
0.049
```

We should see that the proportion of samples with p < .05 is about 5%.