Exercise 11.1 Normal probability plots

The Python code for creating the normal probability plots are given in Listing 1.

(a) The histograms of the raw data are given in Fig. 1. From this plot, it looks like the data in Data/data1.txt could be normally distributed, while the other data sets seem to not follow normal distributions.

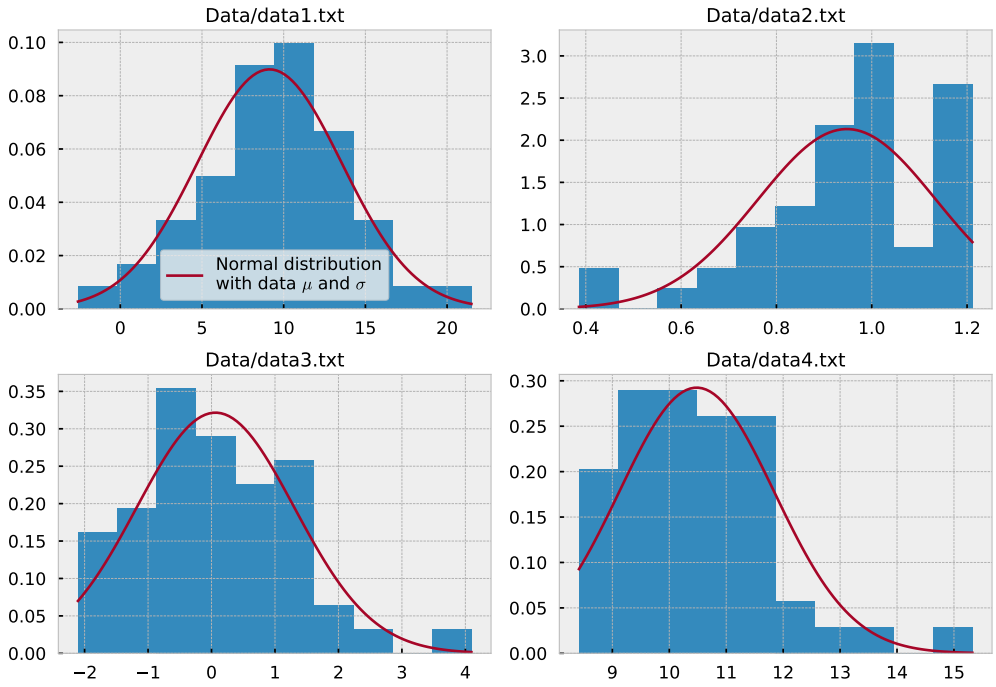

Figure 1: *Histograms for the raw data.*

(b) We calculate the requested values in (i)–(iii) by using the cumulative distribution function and the values in (iv)–(vi) using the percent-point function:

(i) $P(X < 1) = \text{CDF}(1) = 0.841$.

(ii) $P(X < 0) = \text{CDF}(0) = \frac{1}{2}$.

(iii) $P(X < 1) = \text{CDF}(1) = 0.0228$.

(iv) $P(X < \alpha) = 0.10 \implies \alpha = \text{PPF}(0.10) = -1.28$.

(v) $P(X < \alpha) = 0.90 \implies \alpha = \text{PPF}(0.90) = 1.28$.

(vi) $P(X < \alpha) = 0.99 \implies \alpha = \text{PPF}(0.99) = 2.33$.

(c) The normal probability plots are given in Fig. 2. This plot supports our conclusion from point (a) above: the data in Data/data1.txt seems to be normally distributed, while the

other data sets seem to not follow normal distributions. We note that Data/data3.txt is a borderline case: the deviations from a straight line are not too large. However, interpreting Fig. 2 in light of the corresponding histogram in Fig. 1, we still conclude that this data is probably not from a normal distribution. In fact, this data set is obtained from the Student's t-distribution which is expected to approximate a normal distribution for large degrees of freedom.

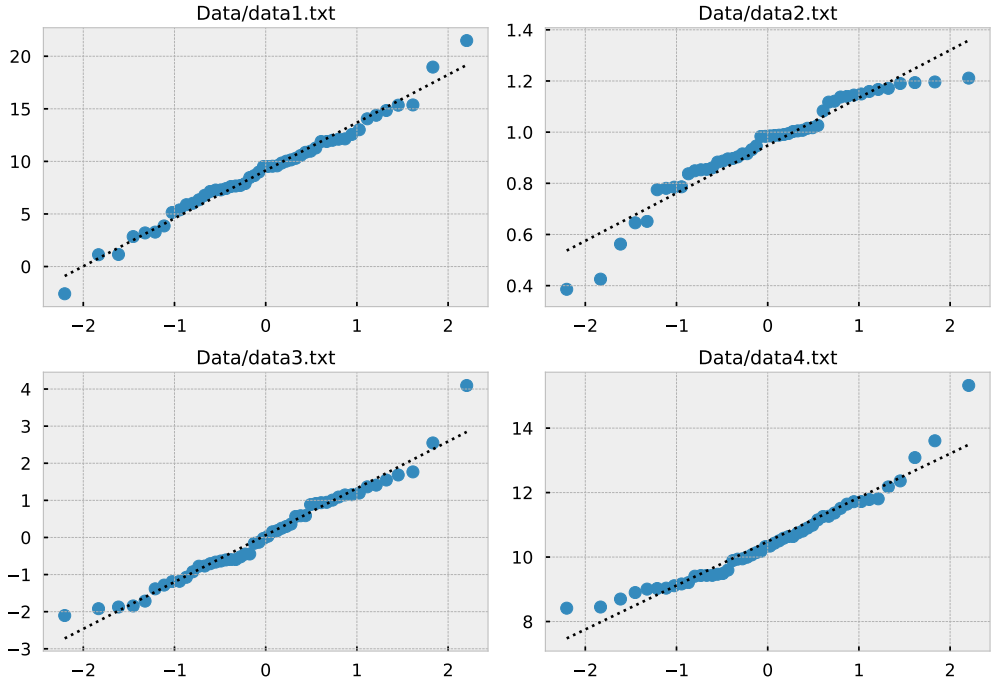

Figure 2: *Normal probability plots for the raw data.*

(d) When checking for the Gumbel distribution, we obtain probability plots as given in Fig. 3. We see here that Data/data4.txt falls on a straight line, while the other data sets show deviations from a straight line. We thus conclude that Data/data4.txt contains data from a Gumbel distribution, while the other data sets do not.

Exercise 11.2 Normal probability plots and experimental design

The normal probability plot of the effects is given in Fig. 4 (see Listing 1 for the Python code used to produce this plot). We see here that 4 factors deviate significantly from the other effects, and a straight line through these points: A, D, BD, and B. We thus conclude that these 4 factors are the important factors in this case.

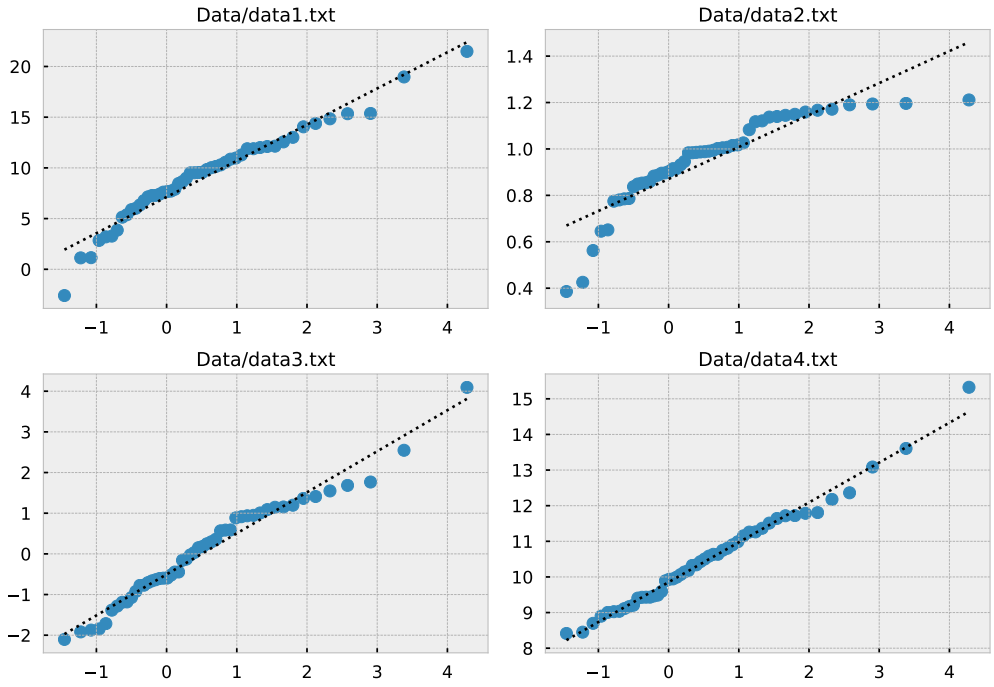

Figure 3: *Probability plots for a Gumbel distribution.*

Exercise 11.3 ANOVA

(i) The significance level, α, is the probability of rejecting the null hypothesis (in this case that all the means are equal) when the null hypothesis is true. Here, we set this value to 0.05 which means that it should be no more than a 5% chance of erroneously concluding that the null hypothesis is false.

(ii) The Python code for performing the ANOVA analysis is given in Listing 2. The main results are summarized in Table 1 and a graphical representation of the raw data is given in Fig. 5. When comparing with the given critical f-value at a significance level of 0.05, we see that this value (2.866) is larger than our calculated f-value: $2.252 < 2.866$. We will then *not* reject the null hypothesis and we conclude that the means are *not* different in this case. We will then recommend to use the lowest concentration as this will probably be cheaper.

The p-value given in Table 1 gives the probability of observing the data we have (or more extreme) given that the null hypothesis is true. This value is close to 10%. Although Fig. 5 indicate that there might be a weak effect when increasing the fertilizer concentration, our test tells us that the probability of observing this by chance is almost 10%, even when the means are the same.

Note: If we were to do a more thorough analysis, we see that two of the points for the concentration 200 g/bu appears to be outliers (see Fig. 5). If we remove these two

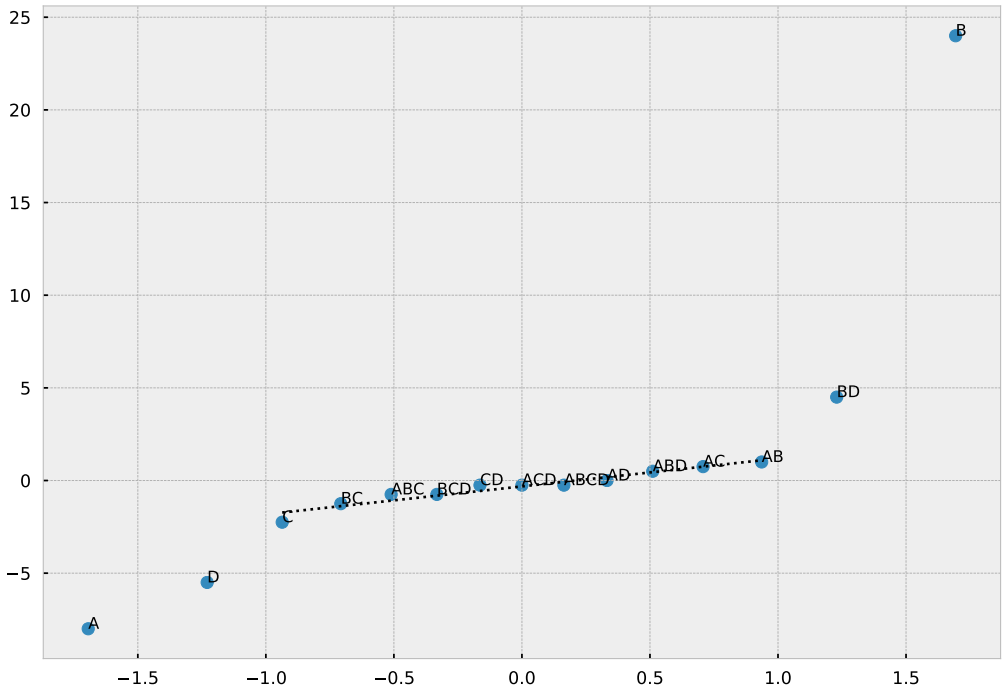

Figure 4: *Normal probability plots for different effects.*

Property	**Value**
SST	758.035
SSA	119.787
SSE	638.248
Degrees of freedom (SSA)	3
Degrees of freedom (SSE)	36
$f = \frac{SSA/3}{SSE/36}$	2.252
p	0.0989

Table 1: *Main ANOVA results for the fertilizer data.*

points, and re-run the ANOVA, we get a new f-value equal to 3.05 and a p-value of 0.04. In this case, we would actually reject the null hypothesis.

(iii) If we select a significance level of 0.10, the critical f-value is 2.243, which is *smaller* than the observed f-value: $2.252 > 2.243$. In this case, we will reject the null hypothesis and find that there is an effect when increasing the fertilizer concentration. We note here that this is very close to the p-value we have calculated in Table 1 (the critical f-value and the calculated one are also very similar). In a real-life situation, we would

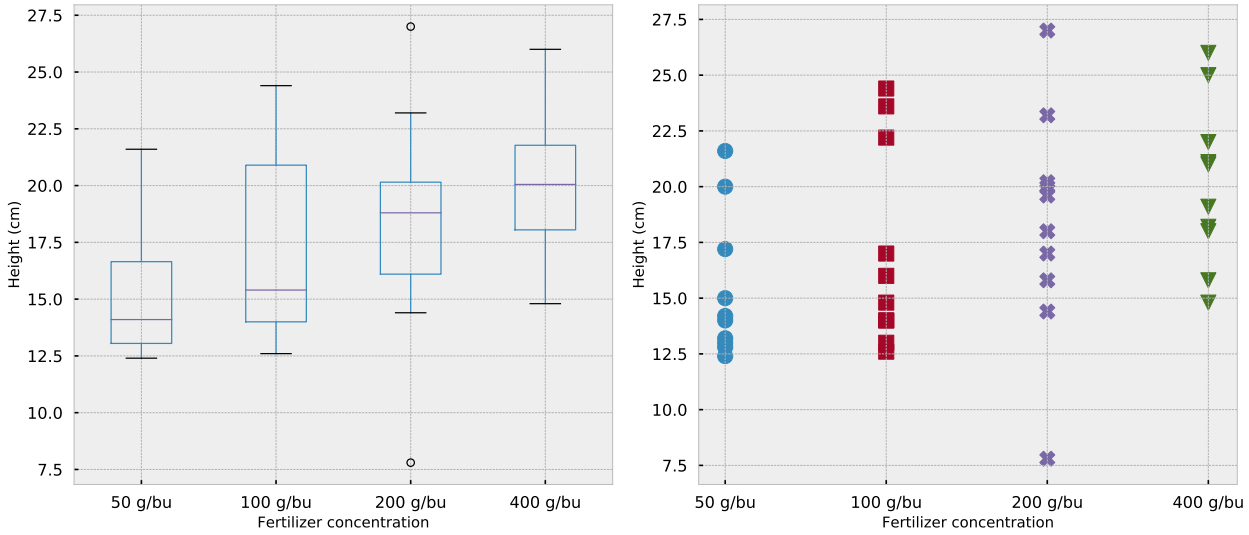

Figure 5: *Box-plot and scatter plot of the fertilizer raw data.*

probably still be skeptical about rejecting the null hypothesis as the effect of increasing the fertilizer concentration seems to be modest, and since we now have a 10% chance of falsely rejecting the null hypothesis.

Exercise 11.4 ANOVA & experimental design

The calculated contrasts, effects, f-values, and p-values are given in Table 2, and the Python code used to calculate these values are given in Listing 3.

Factor	Contrast	Effect	Sum of squares	f-value	p-value
A	15	2.5	18.75	18.75	2.5×10^{-3}
B	-45	-7.5	168.75	168.75	1.2×10^{-6}
AB	-105	-17.5	918.75	918.75	1.5×10^{-6}

Table 2: *Main ANOVA results for the 2^2 factorial experiment, repeated 3 times.*

Comparing the calculated f-values with the given critical value (11.259) we see that all the effects are significant at a 0.01 significance level.

This is also reflected in the calculated p-values. All of these are small, which means that if the effects are *not significant*, then the probability of observing what we have observed is small. Since this probability is small, we thus reject the null hypothesis (the effects are not significant) and conclude that all these effects are significant.

Python solutions

```
from scipy.stats import norm, gumbel_r
import numpy as np
import matplotlib as mpl
from matplotlib import pyplot as plt
mpl.rcParams["savefig.format"] = 'pdf'
plt.style.use(['seaborn-talk', 'bmh'])


def add_trendline(axi, xdata, ydata, **kwargs):
    """Add a trendline to the given axes.

    Parameters
    ----------
    axi : object like :class:`matplotlib.axes.Axes`
        The axis to add the trendline to.
    xdata : object like :class:`pandas.core.series.Series`
        The x-values to add a trendline for.
    ydata : object like :class:`pandas.core.series.Series`
        The y-values to add a trendline for.
    **kwargs : dict, optional
        Additional arguments passed to the plotting method.

    Returns
    -------
    line : object like :class:`matplotlib.lines.Line2D`
        The created line.

    """
    param = np.polyfit(xdata, ydata, 1)
    yhat = np.polyval(param, xdata)
    xpoint = np.array([min(xdata), max(xdata)])
    line, = axi.plot(xpoint, np.polyval(param, xpoint), **kwargs)
    return line


def answer_stats():
    """Answer the statistics questions."""
    for x in (1, 0, -2):
        p = norm.cdf(x)
        print(
            'The probability of observing x <= {} is: {}'.format(x, p)
        )
    for p in (0.10, 0.90, 0.99):
        x = norm.ppf(p)
        print(
            'The probability of observing x <= {} is: {}'.format(x, p)
        )
```

```
def plot_histograms():
    """Plot the raw data as histograms."""
    fig, axes = plt.subplots(constrained_layout=True, nrows=2, ncols=2)
    axes = axes.flatten()
    for i in range(1, 5):
        filei = 'Data/data{}.txt'.format(i)
        datai = np.loadtxt(filei)
        axi = axes[i - 1]
        axi.set_title(filei)
        _, bins, _ = axi.hist(datai, density=True)
        x = np.linspace(min(bins), max(bins), 100)
        y = norm.pdf(x, datai.mean(), datai.std())
        axi.plot(
            x, y,
            label='Normal distribution\n'r'with data $\mu$ and $\sigma$'
        )
        if i == 1:
            axi.legend()
    fig.savefig('histogram', bbox_inches='tight')


def norm_pp(data, distribution=norm):
    """Create x and y values for plotting a normal probability plot."""
    uval = []
    last = len(data)
    for i, _ in enumerate(data):
        if i == 0:
            uval.append(1.0 - (0.5)**(1.0/last))
        elif i == last - 1:
            uval.append((0.5)**(1.0/last))
        else:
            uval.append(((i + 1) - 0.3175) / (last + 0.365))
    xval = distribution.ppf(uval)
    return xval, sorted(data)


def plot_norm_pp(distribution=norm, output='normpp'):
    """Make normal probability plots"""
    fig, axes = plt.subplots(constrained_layout=True, nrows=2, ncols=2)
    axes = axes.flatten()
    for i in range(1, 5):
        filei = 'Data/data{}.txt'.format(i)
        datai = np.loadtxt(filei)
        xdata, ydata = norm_pp(datai, distribution=distribution)
        axi = axes[i - 1]
        axi.set_title(filei)
        axi.scatter(xdata, ydata)
        add_trendline(axi, xdata, ydata, ls=':', color='k')
    fig.savefig(output, bbox_inches='tight')
```

```
def important_effects():
    effects = [
        ('A', -8.00),
        ('B', 24.00),
        ('C', -2.25),
        ('D', -5.50),
        ('AB', 1.00),
        ('AC', 0.75),
        ('AD', 0.00),
        ('BC', -1.25),
        ('BD', 4.50),
        ('CD', -0.25),
        ('ABC', -0.75),
        ('ACD', -0.25),
        ('BCD', -0.75),
        ('ABCD', -0.25),
        ('ABD', 0.50),
    ]
    effects = sorted(effects, key=lambda val: val[1])
    data = [i[1] for i in effects]
    xdata, ydata = norm_pp(data)
    fig, ax1 = plt.subplots(constrained_layout=True)
    ax1.scatter(xdata, ydata)
    add_trendline(ax1, xdata[2:-2], ydata[2:-2], ls=':', color='k')
    for xi, yi, txt in zip(xdata, ydata, effects):
        ax1.annotate(txt[0], (xi, yi), fontsize='large')
    fig.savefig('effects', bbox_layout='tight')


def main():
    plot_histograms()
    answer_stats()
    plot_norm_pp()
    plot_norm_pp(distribution=gumbel_r, output='gumbelpp')

    important_effects()

    plt.show()


if __name__ == '__main__':
    main()
```

Listing 1: *Python code for creating normal probability plots.*

```
import scipy.stats
import numpy as np
import pandas as pd
import matplotlib as mpl
```

```
from matplotlib import pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols
mpl.rcParams["savefig.format"] = 'pdf'
plt.style.use(['seaborn-talk', 'bmh'])


def run_anova(data, alpha=0.05):
    n, k = data.shape  # Rows = Experiments, Column = Treatments
    mean = np.average(data)  # Mean of all data.
    SST = np.sum((data - mean)**2)
    print('SST =', SST)
    means = np.average(data, axis=0)  # Mean of columns.
    print('Means', means)
    SSA = n * np.sum((means - mean)**2)
    print('SSA =', SSA)
    SSE = SST - SSA
    print('SSE =', SST - SSA)
    df1 = (k - 1)
    df2 = k * (n - 1)
    s1 = SSA / df1
    s = SSE / df2
    f = s1 / s
    print('s1 = ', s1)
    print('s = ', s)
    print('f =', f)
    f_critical = scipy.stats.f.ppf(1 - alpha, df1, df2)
    print('f_critical =', f_critical)
    pval = 1 - scipy.stats.f.cdf(f, df1, df2)
    print('pval =', pval)
    print('pval < alpha:', pval < alpha)
    print('f > f_critical:', f > f_critical)


def plot_raw_data(ax1, data, xlabels):
    _, columns = data.shape
    markers = ['o', 's', 'X', 'v', 'D']
    for i in range(columns):
        column = data[:, i]
        x = np.ones_like(column) * (i + 1)
        ax1.scatter(x, column, s=150, marker=markers[i])
    ax1.set(xlabel='Fertilizer concentration', ylabel='Height (cm)')
    ax1.set_xticks(range(1, columns + 1))
    ax1.set_xticklabels(xlabels)


def main():
    data = pd.read_csv('Data/fertilizer.txt')
    data_mat = data.values
    # Plot raw data:
```

```
    figbox, (axbox, axscat) = plt.subplots(
        constrained_layout=True,
        ncols=2,
        figsize=(14, 6)
    )
    data.boxplot(ax=axbox)
    axbox.set_xlabel('Fertilizer concentration')
    axbox.set_ylabel('Height (cm)')
    plot_raw_data(axscat, data_mat, data.columns)
    figbox.savefig('fertilizer', bbox_inches='tight')

    # Run anova "by hand":
    run_anova(data_mat)

    # Let us check the results using scipy:
    columns = [data_mat[:, i] for i in range(data_mat.shape[1])]
    fval, pval = scipy.stats.f_oneway(*columns)
    print('\nScipy results:')
    print('Scipy f =', fval)
    print('Scipy p =', pval)

    # Let us also check the results using statsmodels:
    # We first have to reorganize the data for statsmodels:
    data_melt = pd.melt(data.reset_index(), id_vars=['index'],
                        value_vars=data.columns)
    model = ols('value ~ C(variable)', data=data_melt).fit()
    table = sm.stats.anova_lm(model, typ=2)
    print('\nStatsmodels results:')
    print(table)

    plt.show()


if __name__ == '__main__':
    main()
```

Listing 2: *Python code for running ANOVA.*

```
from itertools import chain, combinations
from functools import reduce
import operator
import scipy.stats
import numpy as np


def powerset(iterable):
    "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)"
    s = list(iterable)
    return chain.from_iterable(combinations(s, r) for r in range(len(s)+1))
```

```
def main():
    k = 3  # 3 repeated experiments
    N = 2  # 2 factors.
    data = [
        ['(1)', 9, 10, 11],
        ['a', 30, 31, 29],
        ['b', 19, 20, 21],
        ['ab', 5, 6, 4],
    ]

    signs = {}
    for key in ('a', 'b'):
        signs[key] = np.array([1 if key in row[0] else -1 for row in data])

    # Obtain contrasts:
    contrasts = {}
    for factors in powerset(('a', 'b')):
        if not factors:
            continue
        factor_name = ''.join(factors)
        factor_sign = reduce(operator.mul, [signs[i] for i in factors])
        contrasts[factor_name] = 0
        for sign, experiment in zip(factor_sign, data):
            contrasts[factor_name] += sign * sum(experiment[1:])
    print('Contrasts:', contrasts)

    effects = {}
    # Calculate effects:
    for factor, contrast in contrasts.items():
        effects[factor] = contrast / (k * 2**(N - 1))
    print('Effects:', effects)

    # Calculate sum of squares:
    sum_squares = {}
    for factor, contrast in contrasts.items():
        sum_squares[factor] = contrast*contrast / (k * 2**N)
    print('Sum of squares:', sum_squares)

    # Calculate total sum of squares:
    data_values = np.array([i[1:] for i in data])
    mean = data_values.mean()
    SST = np.sum((data_values - mean)**2)
    print('SST =', SST)

    SSE = SST - sum([val for _, val in sum_squares.items()])
    print('SSE =', SSE)
    df_sse = 2**2 * (k - 1)
    print('Degrees of freedom SSE:', df_sse)
    sigma = SSE / df_sse
```

```
    # Calculate f-values:
    print()
    for factor, ssi in sum_squares.items():
        fval = ssi / sigma
        pval = 1 - scipy.stats.f.cdf(fval, 1, df_sse)
        print('ANOVA for: {}'.format(factor))
        print('\tf = {}'.format(fval))
        print('\tp = {}'.format(pval))

    # Print out some critical f-values at different significance levels:
    print('\nCritical f-values:')
    for alpha in (0.001, 0.01, 0.05, 0.1):
        f_critical = scipy.stats.f.ppf(1 - alpha, 1, df_sse)
        print(
            '\tAt alpha {:5.3f}: f-critical = {:6.3f}'.format(
                alpha, f_critical
            )
        )


if __name__ == '__main__':
    main()
```

Listing 3: *Python code for running ANOVA for experimental design data.*