```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy
from uncertainties import ufloat
import toolkit as tk

# Constants
# Note that we are measuring a 3x3 grid which is where the multiplication
# by 3 comes from
GRID_SIZE = 1 # mm
GRID_SIZE = GRID_SIZE*3

CALIPER_UNCERT = 0.01

# Measurement Uncertainty remains the same
GRID_SIZE_u = 0.5 #mm uncert

OFFSETS = {
    "LENS":59.95,
    'LENS-T':25.42,
    "IMG":59.82,
    'LAMP':90.74,
    'APP':59.82
    }


def thin_lens_eqn(p, q, f, fp, bp):
    return 1/(p-fp) + 1/(q-bp)


def thin_lens_eqn2(p, f, fp, bp):
    #numer = (fp*bp - p*bp + f*bp + f*fp - fp)/(f - p + fp)
    #denom = f - p + fp

    return (fp*bp - p*bp + f*bp + f*fp - fp)/(f - p + fp)

def thin_lens_eqn3(p, f, fp, bp):
    return ((1/f)-(1/(p-fp)))**(-1) + bp

def thin_lens_eqn4(p, f_inv, fp, bp):
    return f_inv - 1/(p+fp) + bp

def thin_lens_eqn5(p, finv, fp, bp):
    return (finv-(1/(p-fp)))**(-1) + bp

def load_data(path):
    df = pd.read_csv(path)


    return df

def analyze_old(df, grid_size):
    df_mean = df.mean()
    df_std = df.std()

    img = ufloat(df_mean['image'], df_std['image'])
    obj = ufloat(df_mean['lens'], df_std['lens'])
    lens = ufloat(df_mean['source'], df_std['source'])

    #print(img, obj, lens)
```

```python
    #mag_uncert = ufloat(df_mean['gridsize'], df_std['gridsize'])
    #gridsize_uncert = ufloat(GRID_SIZE, GRID_SIZE_u)

    p = np.abs(lens - obj)
    q = np.abs(lens - img)

    #print(p,q)

    magnification = -(q/p)

    #magnification = mag_uncert/gridsize_uncert

    output = np.array(
        [p,
         q,
         magnification])

    return output

def analyze(df, grid_size):

    df['Uncert'] = pd.Series([CALIPER_UNCERT for _ in range(len(df))])

    print(df)


    img = [ufloat(i,CALIPER_UNCERT) for i in df['image'].to_numpy()]
    lens = [ufloat(i,CALIPER_UNCERT) for i in df['lens'].to_numpy()]
    src = [ufloat(i,CALIPER_UNCERT) for i in df['source'].to_numpy()]

    img = df['image'].to_numpy()
    lens = df['lens'].to_numpy()
    src =  df['source'].to_numpy()

    # Adding offsets
    img = img + 0.5*OFFSETS['IMG']*0.1
    lens = lens + 0.5*OFFSETS['LENS']*0.1 + OFFSETS['LENS-T']*0.1
    src = src - 0.5*OFFSETS['APP']

    #print(img, obj, lens)

    #mag_uncert = ufloat(df_mean['gridsize'], df_std['gridsize'])
    #gridsize_uncert = ufloat(GRID_SIZE, GRID_SIZE_u)

    p = np.abs(lens - src)
    q = np.abs(lens - img)

    #print(p,q)

    m = df['gridsize'].div(3)

    magnification = -(q/p)

    #magnification = mag_uncert/gridsize_uncert

    output =(p.tolist(),
             q.tolist(),
             m.tolist())

    return output
```

```python
def fit_thin_lens_old(pdata, qdata, mdata):
    #print(pdata[2].n)

    pdata_n = np.array([(p.n) for p in pdata])
    #pdata_u = np.array([p.s for p in pdata])

    pdata_u = np.array([0.5, 0.5, 0.5])


    qdata_n = np.array([q.n for q in qdata])
    qdata_u = np.array([q.s for q in qdata])


    # Encountering Division by zero in degrees of freedom
    data = tk.curve_fit_data(pdata_n, qdata_n, 'custom', uncertainty=qdata_u,
                       res=True, chi=True, uncertainty_x=pdata_u,
                       model_function_custom=thin_lens_eqn2)
    #print(data)

    print('popt:', data['popt'])

    plt.scatter(qdata_n, pdata_n)


    meta = {
        'title':'verification of thin lens',
        'x-label':'p values',
        'y-label':'q values'
        }
    tk.quick_plot_residuals(qdata_n, pdata_n, data['graph-horz'],
                            data['graph-vert'], data['residuals'],
                            uncertainty=qdata_u,meta=meta)


    data = tk.curve_fit_data(pdata_n, qdata_n, 'custom', uncertainty=qdata_u,
                       res=True, chi=True, uncertainty_x=pdata_u,
                       model_function_custom=thin_lens_eqn2)

    plt.show()

def fit_thin_lens(pdata, qdata, mdata):
    #print(pdata[2].n)

    pdata_n = np.array([p for p in pdata])
    #pdata_u = np.array([p.s for p in pdata])

    pdata_u = np.array([CALIPER_UNCERT]*len(pdata))


    qdata_n = np.array([q for q in qdata])
    qdata_u = np.array([CALIPER_UNCERT]*len(qdata))


    # Encountering Division by zero in degrees of freedom
    data = tk.curve_fit_data(pdata_n, qdata_n, fit_type='custom',
                             uncertainty=qdata_u,
                             res=True, chi=True,
                             model_function_custom=thin_lens_eqn5,)
    #print(data)

    print('popt:', data['popt'])
```

```python
    #plt.scatter(qdata_n, pdata_n)

    meta = {'title':'Verification of thin lens',
            'xlabel':'p values',
            'ylabel':'q values',
            'data-label':'Measurements',
            'fit-label':r'$q=\left(\frac{1}{f}-\frac{1}{p-fp}\right)^{-1} + bp$',
            'loc':'upper right',
            'save-name':'pq-ideal'}

    plt.figure()

    xdata = np.arange(29, 70, 0.1)
    ydata = thin_lens_eqn5(xdata, *data['popt'])
    tk.quick_plot_residuals(qdata_n, pdata_n, xdata,
                            ydata, data['residuals'],
                            uncertainty=qdata_u,meta=meta)

    plt.show()

    print(data['popt'], data['pcov'])


def fit_mag(pdata, qdata, mdata):
    mideal = -qdata/pdata

    print(mdata)
    uncert = len(mideal)*[CALIPER_UNCERT]

    data = tk.curve_fit_data(mideal, mdata, fit_type='linear-int',
                             uncertainty=uncert, res=True, chi=True)

    meta = {'title': 'Predicted Magnification vs. Actual',
            'xlabel':'Predicted Magnification',
            'ylabel':'Actual Magnification',
            'data-label':'Measurements',
            'fit-label':r'$M=-\frac{Q}{P}$',
            'loc':'upper right',
            'save-name':'mag-ideal'}
    plt.figure()
    tk.quick_plot_residuals(mideal, mdata, data['graph-horz'],
                            data['graph-vert'], data['residuals'],
                            uncertainty=uncert,meta=meta)

    print(data['popt'], data['pcov'])

    plt.show()

if __name__ == '__main__':
    df = load_data('../Data/ExpA-2025.01.24-1.csv')
    df_11 = df[df['lens-code']==1.1]
    df_12 = df[df['lens-code']==1.2]
    df_13 = df[df['lens-code']==1.3]
    df_14 = df[df['lens-code']==1.4 ]


    #print(df)
    out_11 = analyze(df_11, GRID_SIZE)
    out_12 = analyze(df_12, GRID_SIZE)
    out_13 = analyze(df_13, GRID_SIZE)
    out_14 = analyze(df_14, GRID_SIZE)
```

```
xtest = np.arange(27,50, 0.1)
ytest = thin_lens_eqn3(xtest, f=5.6, fp=14.6, bp=-5.2)

plt.figure()
plt.plot(xtest, ytest)
plt.show()

pdata = np.array(out_11[0] + out_12[0] + out_13[0] + out_14[0])
qdata = np.array(out_11[1] + out_12[1] + out_13[1] + out_14[1])
mdata = np.array(out_11[2] + out_12[2] + out_13[2] + out_14[2])

fit_mag(pdata, qdata, mdata)
fit_thin_lens(pdata, qdata, mdata)

df2 = load_data('../Data/ExpA-2025.01.31-1.csv')
df_21 = df2[df2['lens-code']==2.1]
out_21 = analyze(df_21, GRID_SIZE)
pdata = np.array(out_21[0])
qdata = np.array(out_21[1])
mdata = np.array(out_21[2])
fit_mag(pdata, qdata, mdata)
fit_thin_lens(pdata, qdata, mdata)


df3 = load_data('../Data/ExpA-2025.01.31-1.csv')
df_31 = df3[df3['lens-code']==3.1]
out_31 = analyze(df_31, GRID_SIZE)
pdata = np.array(out_31[0])
qdata = np.array(out_31[1])
mdata = np.array(out_31[2])
fit_mag(pdata, qdata, mdata)
fit_thin_lens(pdata, qdata, mdata)
```