Project 4: Hacking Food & Nutrition

Team Zilberman

Target States in India: Tamil Nadu, West Bengal
Key: Comparison across two states

Goals:

- In this project we will identify the food demand systems and nutritional systems within the populations of two regions in **India: Bengal (West Bengal) and Tamil Nadu**. Both of these regions suffer from nutritional inadequacy as there has been recently an emphasis on the quantity of food produced (large scale cash crops), rather than diverse nutritional quality. We will assess which nutrients are most lacking in each population and propose policies that will foster a healthier and more sustainable food supply, all while considering food prices, household budgets, and other household characteristics within these populations.

Table of contents

Import Data Libraries

In [2]:
```
!pip install -r requirements.txt
import cfe

cfe.Result?
import pandas as pd
from cfe.df_utils import to_dataframe

import ipywidgets
from ipywidgets import interactive, fixed, interact, Dropdown
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings

import fooddatacentral as fdc
```

```
Requirement already satisfied: CFEDemands>=0.4.1 in /opt/conda/lib/python3.9/site-packag
es (from -r requirements.txt (line 5)) (0.4.1)
Requirement already satisfied: gspread>=5.0.1 in /opt/conda/lib/python3.9/site-packages
(from -r requirements.txt (line 8)) (5.3.2)
Requirement already satisfied: matplotlib>=3.3.4 in /opt/conda/lib/python3.9/site-packag
es (from -r requirements.txt (line 11)) (3.4.3)
```

```
Collecting numpy>=1.22.2
  Using cached numpy-1.22.3-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1
6.8 MB)
Requirement already satisfied: oauth2client>=4.1.3 in /opt/conda/lib/python3.9/site-pack
ages (from -r requirements.txt (line 18)) (4.1.3)
Collecting pandas>=1.4.1
  Using cached pandas-1.4.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1
1.7 MB)
Collecting plotly>=5.5.0
  Using cached plotly-5.7.0-py2.py3-none-any.whl (28.8 MB)
Requirement already satisfied: eep153_tools>=0.11 in /opt/conda/lib/python3.9/site-packa
ges (from -r requirements.txt (line 28)) (0.11)
Requirement already satisfied: gnupg in /opt/conda/lib/python3.9/site-packages (from -r
requirements.txt (line 29)) (2.3.1)
Requirement already satisfied: ConsumerDemands in /opt/conda/lib/python3.9/site-packages
(from -r requirements.txt (line 31)) (0.3.dev0)
Requirement already satisfied: google-auth>=1.12.0 in /opt/conda/lib/python3.9/site-pack
ages (from gspread>=5.0.1->-r requirements.txt (line 8)) (2.6.2)
Requirement already satisfied: google-auth-oauthlib>=0.4.1 in /opt/conda/lib/python3.9/s
ite-packages (from gspread>=5.0.1->-r requirements.txt (line 8)) (0.4.5)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.9/site-packages (f
rom matplotlib>=3.3.4->-r requirements.txt (line 11)) (0.11.0)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.9/site-packages
(from matplotlib>=3.3.4->-r requirements.txt (line 11)) (8.3.2)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.9/site-package
s (from matplotlib>=3.3.4->-r requirements.txt (line 11)) (3.0.7)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.9/site-pac
kages (from matplotlib>=3.3.4->-r requirements.txt (line 11)) (2.8.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.9/site-packag
es (from matplotlib>=3.3.4->-r requirements.txt (line 11)) (1.4.2)
Requirement already satisfied: rsa>=3.1.4 in /opt/conda/lib/python3.9/site-packages (fro
m oauth2client>=4.1.3->-r requirements.txt (line 18)) (4.8)
Requirement already satisfied: httplib2>=0.9.1 in /opt/conda/lib/python3.9/site-packages
(from oauth2client>=4.1.3->-r requirements.txt (line 18)) (0.20.4)
Requirement already satisfied: six>=1.6.1 in /opt/conda/lib/python3.9/site-packages (fro
m oauth2client>=4.1.3->-r requirements.txt (line 18)) (1.16.0)
Requirement already satisfied: pyasn1>=0.1.7 in /opt/conda/lib/python3.9/site-packages
(from oauth2client>=4.1.3->-r requirements.txt (line 18)) (0.4.8)
Requirement already satisfied: pyasn1-modules>=0.0.5 in /opt/conda/lib/python3.9/site-pa
ckages (from oauth2client>=4.1.3->-r requirements.txt (line 18)) (0.2.8)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.9/site-packages (f
rom pandas>=1.4.1->-r requirements.txt (line 23)) (2021.1)
Requirement already satisfied: tenacity>=6.2.0 in /opt/conda/lib/python3.9/site-packages
(from plotly>=5.5.0->-r requirements.txt (line 26)) (8.0.1)
Requirement already satisfied: psutil>=1.2.1 in /opt/conda/lib/python3.9/site-packages
(from gnupg->-r requirements.txt (line 29)) (5.9.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /opt/conda/lib/python3.9/site-p
ackages (from google-auth>=1.12.0->gspread>=5.0.1->-r requirements.txt (line 8)) (5.0.0)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /opt/conda/lib/python3.9/site
-packages (from google-auth-oauthlib>=0.4.1->gspread>=5.0.1->-r requirements.txt (line
8)) (1.3.1)
Requirement already satisfied: requests>=2.0.0 in /opt/conda/lib/python3.9/site-packages
(from requests-oauthlib>=0.7.0->google-auth-oauthlib>=0.4.1->gspread>=5.0.1->-r requirem
ents.txt (line 8)) (2.26.0)
Requirement already satisfied: oauthlib>=3.0.0 in /opt/conda/lib/python3.9/site-packages
(from requests-oauthlib>=0.7.0->google-auth-oauthlib>=0.4.1->gspread>=5.0.1->-r requirem
ents.txt (line 8)) (3.2.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/python3.9/site-pa
ckages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthlib>=0.4.1->gsp
read>=5.0.1->-r requirements.txt (line 8)) (1.25.7)
Requirement already satisfied: idna<4,>=2.5; python_version >= "3" in /opt/conda/lib/pyt
hon3.9/site-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthl
ib>=0.4.1->gspread>=5.0.1->-r requirements.txt (line 8)) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.9/site-packa
ges (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthlib>=0.4.1->gsprea
d>=5.0.1->-r requirements.txt (line 8)) (2019.11.28)
```

```
Requirement already satisfied: charset-normalizer~=2.0.0; python_version >= "3" in /opt/
conda/lib/python3.9/site-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->googl
e-auth-oauthlib>=0.4.1->gspread>=5.0.1->-r requirements.txt (line 8)) (2.0.0)
Installing collected packages: numpy, pandas, plotly
  Attempting uninstall: numpy
    Found existing installation: numpy 1.21.5
    Uninstalling numpy-1.21.5:
      Successfully uninstalled numpy-1.21.5
  Attempting uninstall: pandas
    Found existing installation: pandas 1.3.5
    Uninstalling pandas-1.3.5:
      Successfully uninstalled pandas-1.3.5
  Attempting uninstall: plotly
    Found existing installation: plotly 5.2.1
    Uninstalling plotly-5.2.1:
      Successfully uninstalled plotly-5.2.1
ERROR: After October 2020 you may experience errors when installing or updating package
s. This is because pip will change the way that it resolves dependency conflicts.

We recommend you use --use-feature=2020-resolver to test your packages with the new reso
lver before it becomes the default.

tensorflow 2.6.3 requires h5py~=3.1.0, but you'll have h5py 3.3.0 which is incompatible.
tensorflow 2.6.3 requires numpy~=1.19.2, but you'll have numpy 1.22.3 which is incompati
ble.
tensorflow 2.6.3 requires six~=1.15.0, but you'll have six 1.16.0 which is incompatible.
tensorboard 2.6.0 requires google-auth<2,>=1.6.3, but you'll have google-auth 2.6.2 whic
h is incompatible.
pysal 2.5.0 requires urllib3>=1.26, but you'll have urllib3 1.25.7 which is incompatibl
e.
pynwb 1.5.1 requires h5py<3,>=2.9, but you'll have h5py 3.3.0 which is incompatible.
pynwb 1.5.1 requires hdmf<3,>=2.5.6, but you'll have hdmf 2.4.0 which is incompatible.
pynwb 1.5.1 requires numpy<1.21,>=1.16, but you'll have numpy 1.22.3 which is incompatib
le.
pandas 1.4.2 requires python-dateutil>=2.8.1, but you'll have python-dateutil 2.8.0 whic
h is incompatible.
numba 0.55.1 requires numpy<1.22,>=1.18, but you'll have numpy 1.22.3 which is incompati
ble.
hdmf 2.4.0 requires h5py<3,>=2.9, but you'll have h5py 3.3.0 which is incompatible.
hdmf 2.4.0 requires jsonschema<4,>=2.6.0, but you'll have jsonschema 4.4.0 which is inco
mpatible.
hdmf 2.4.0 requires numpy<1.19.4,>=1.16, but you'll have numpy 1.22.3 which is incompati
ble.
fenics-dolfin 2019.1.0 requires pybind11==2.2.4, but you'll have pybind11 2.8.1 which is
 incompatible.
fancyimpute 0.6.0 requires keras==2.4.3, but you'll have keras 2.6.0 which is incompatib
le.
fancyimpute 0.6.0 requires numpy==1.19.5, but you'll have numpy 1.22.3 which is incompat
ible.
fancyimpute 0.6.0 requires scipy==1.6.3, but you'll have scipy 1.7.3 which is incompatib
le.
fancyimpute 0.6.0 requires tensorflow==2.5, but you'll have tensorflow 2.6.3 which is in
compatible.
csaps 1.0.4 requires numpy<1.21.0,>=1.11.0, but you'll have numpy 1.22.3 which is incomp
atible.
csaps 1.0.4 requires scipy<1.7.0,>=1.0.0, but you'll have scipy 1.7.3 which is incompati
ble.
allensdk 2.12.2 requires aiohttp==3.7.4, but you'll have aiohttp 3.8.1 which is incompat
ible.
allensdk 2.12.2 requires h5py<3.0.0,>=2.8, but you'll have h5py 3.3.0 which is incompati
ble.
allensdk 2.12.2 requires jinja2<2.12.0,>=2.7.3, but you'll have jinja2 3.1.1 which is in
compatible.
allensdk 2.12.2 requires matplotlib<3.4.3,>=1.4.3, but you'll have matplotlib 3.4.3 whic
h is incompatible.
allensdk 2.12.2 requires nest-asyncio==1.2.0, but you'll have nest-asyncio 1.5.4 which i
```

```
s incompatible.
allensdk 2.12.2 requires numpy<1.19.0,>=1.15.4, but you'll have numpy 1.22.3 which is in
compatible.
allensdk 2.12.2 requires pandas<=0.25.3,>=0.25.1, but you'll have pandas 1.4.2 which is
 incompatible.
allensdk 2.12.2 requires scikit-image<0.17.0,>=0.14.0, but you'll have scikit-image 0.1
8.3 which is incompatible.
allensdk 2.12.2 requires xarray<0.16.0, but you'll have xarray 0.19.0 which is incompati
ble.
Successfully installed numpy-1.22.3 pandas-1.4.2 plotly-5.7.0
Missing dependencies for OracleDemands.
```

[A] Choice of Dataset

We acquired our data from the Indian National Sample Survey (NSS). These original parque files contain data from a very large pool of households from 35 states; the following parts establish dataframes for our choosen Bengal and Tamil Nadu population.

The raw data processing steps are omitted from this notebook for the sake of conciseness. Throughout this project, we identified and fixed some significicant data issue with the raw files from project 3:

- unit of quantity not standardized
- quantity listed in kg and liters are in fact in grams

These would create huge discrepency and undermine the credibility of our estimation. Upon fixing these issues, we are going to start project 4 with directly reading the datasets saved with the estimation results using methodology adapted from project 3.

Since we are examining two states, we have to run two sets of identical code of all deliverables for each state.

For Tamil Nadu

A. [A] Estimate Demand System

An instance `r` of `cfe.Result` can be made persistent with `r.to_dataset('my_result.ds')`, which saves the instance “on disk” in NetCDF format, and can be loaded using `cfe.from_dataset`. We use this method below to load data and demand system estimated from the NSS Tamil Nadu data:

In [3]:
```
#reading results saved as a ds
r = cfe.from_dataset('./tamil_nadu_final_result.ds',engine='netcdf4')
r
```

Out[3]: xarray.Result

▸ Dimensions: (**i**: 90, **k**: 19, **t**: 1, **m**: 1, **j**: 6647, **kp**: 19)

▾ Coordinates:

i	(i)	object	'apple' ... 'wheat/atta - other ...

k	(k)	object	'Males 0-1' ... 'log Hsize'
t	(t)	int64	1
m	(m)	int64	1
j	(j)	object	'457101101' ... '709982301'
kp	(kp)	object	'Males 0-1' ... 'log Hsize'

▶ Data variables: (20)

▶ Attributes: (10)

Interpreting Parameters

α:

higher α, larger share in total food expenditure

- more luxry items, such as cooked meals and liquor, constitute a higher proportion in food expenditure
- goods like spices (tumeric, salt, chillies, ginger) intuitively have smaller alphas

In [7]:
```
# alpha sorted in descending order
r.get_alpha(as_df=True).dropna().sort_values(ascending=False)
```

Out[7]:
```
i
cooked meals                    5.530392
foreign liquor or refined liquor    5.502684
lpg                             5.120414
milk: liquid                    5.102567
cigarettes                      5.072392
                                  ...
chillis (green)                 1.406527
salt                            1.290151
matches                         1.162702
ginger                          1.148000
oilseeds                        1.126437
Name: alpha, Length: 90, dtype: float64
```

β:

Income elasticity parameter

- how sensitive demand for a good is compared to changes in other economic factors, such as price or income
- higher beta, more elastic, more demanded when food budget is higher

In [9]:
```
r.get_beta(as_df=True).dropna().sort_values(ascending=False)
```

Out[9]:
```
i
cashewnut                       0.557757
ghee                            0.499771
electricity                     0.448160
carrot                          0.412492
raisin (kishmish, monacca etc.)  0.411014
                                  ...
kerosene-pds                    0.017745
matches                        -0.020537
pan : leaf                     -0.031982
rice - P.D.S.                  -0.062167
```

```
firewood & chips                  -0.106539
Name: beta, Length: 90, dtype: float64
```

δ:

Effect of household characteristic on demand

In [10]:
```
to_dataframe(r.delta).unstack('k')
```

Out[10]:

k i	Males 0-1	Males 1-5	Males 5-10	Males 10-15	Males 15-20	Males 20-30	Males 30-50	Males 50-60	Males 60-100	Fe
apple	0.073193	-0.001645	0.052724	0.028196	-0.010756	0.026764	0.060359	0.093082	0.088612	0.0
arhar (tur)	0.027373	-0.040300	-0.042641	-0.018622	0.037128	0.049466	0.134743	0.113700	0.090923	-0.0
banana	-0.072368	-0.039648	-0.011825	-0.020688	-0.035252	0.017445	0.134937	0.150032	0.107098	-0.0
besan	-0.249525	-0.034396	-0.041101	0.010926	0.045697	-0.003774	0.053424	0.040408	0.055361	0.0
black pepper	-0.116639	-0.096475	-0.072696	-0.046868	-0.020677	-0.003834	0.065494	0.034353	0.016814	-0.2
...	...	...	...	...	...	...	...	...	...	
tomato	-0.044180	-0.094050	-0.079028	-0.075340	-0.064174	-0.061953	0.011734	0.005607	-0.029408	-0.1
turmeric	-0.054917	-0.094117	-0.066273	-0.058445	0.004203	-0.009018	0.046495	0.037282	0.035982	-0.0
urd	0.016715	-0.070871	-0.085657	-0.069282	-0.040767	-0.043362	0.101467	0.106081	0.116233	-0.1
wheat/atta - P.D.S.	0.036509	-0.030420	-0.027166	-0.038756	0.005098	-0.016960	0.028825	0.052745	0.071927	-0.0
wheat/atta - other sources	0.186780	0.014527	0.018736	-0.012288	0.054559	0.045337	0.212971	0.233055	0.251153	-0.1

90 rows × 19 columns

The triple of parameters (α, β, δ) completely describes the demand system and the corresponding utility function (over the goods we observe).

Demands

As mentioned above, we've estimated the parameters of a Frischian demand system (demands that depend on prices and the households marginal utility of expenditures). But we can *compute* the corresponding Marshallian (depends on prices and budget) or Hicksian (depends on prices and the level of utility) demands for this same population, using the `cfe.Result.demands` method.

Let's compute Marshallian demands. Start with a choice of budget x and prices.

In [14]:
```
t=1
m=1

x = r.get_predicted_expenditures().sum('i')
median_x = x.where(x>0).sel(t=t,m=m).median('j') # Budget (median household)

# Note selection of prices for Tamil Nadu
p = r.prices.sel(t=t,m=m).fillna(1).copy()
```

```
p.to_dataframe().fillna(1).squeeze()
p_df = p.to_dataframe().fillna(1).squeeze()
```

We have check the reliability of our estimated prices with respect to actual market price.

In [12]:
```
# showing prices for all goods in descending order
with pd.option_context('display.max_rows', None,):
    print(p_df.sort_values(by = 'prices', ascending=False))
```

```
                                                  t  m      prices
i
foreign liquor or refined liquor                  1  1  565.359021
coffee: powder                                    1  1  531.817776
cashewnut                                         1  1  529.254976
black pepper                                      1  1  405.984789
ghee                                              1  1  378.457671
goat meat                                         1  1  358.042342
tea : leaf                                        1  1  314.452176
raisin (kishmish, monacca etc.)                   1  1  287.350566
jeera                                             1  1  237.175981
curry powder                                      1  1  198.302710
turmeric                                          1  1  188.681783
chips                                             1  1  172.302859
pickles                                           1  1  165.711992
other spices                                      1  1  160.651874
dates                                             1  1  142.064384
chicken                                           1  1  135.456507
fish ( fresh )                                    1  1  131.160388
apple                                             1  1  126.592908
dry chillies                                      1  1  125.928280
tamarind                                          1  1  111.416584
dhania                                            1  1  105.911877
groundnut oil                                     1  1   99.496305
garlic                                            1  1   95.371504
moong                                             1  1   82.458173
groundnut                                         1  1   81.099422
oilseeds                                          1  1   79.016807
refined oil [sunflower, soyabean, saffola, etc.]  1  1   77.523339
sewai, noodles                                    1  1   76.612419
other pulse products                              1  1   74.382986
gram products                                     1  1   72.666643
gram (whole)                                      1  1   69.941371
gram (split)                                      1  1   67.976129
besan                                             1  1   67.844577
curd                                              1  1   65.326598
grapes                                            1  1   64.819892
eggs                                              1  1   63.694572
ginger                                            1  1   63.482791
other pulses                                      1  1   63.095061
bread (bakery)                                    1  1   62.300180
kerosene-other sources                            1  1   54.006825
urd                                               1  1   53.186266
arhar (tur)                                       1  1   52.544116
edible oil (others)                               1  1   52.293983
gur                                               1  1   51.243586
peas-pulses                                       1  1   50.829884
peas-vegetables                                   1  1   41.066854
wheat/atta - other sources                        1  1   37.774307
sugar - other sources                             1  1   36.459964
mango                                             1  1   36.088399
suji, rawa                                        1  1   34.546315
chillis (green)                                   1  1   34.257333
french beans and barbati                          1  1   33.761161
carrot                                            1  1   29.937252
maida                                             1  1   29.622903
```

```
lpg                                                1  1   28.822488
tea : cups                                         1  1   28.498240
rice - other sources                               1  1   27.526789
milk: liquid                                       1  1   27.062438
guava                                              1  1   27.000550
cooked meals                                       1  1   25.038579
ragi & products                                    1  1   24.998811
parwal / patal                                     1  1   23.348604
lemon                                              1  1   23.200959
cauliflower                                        1  1   22.848307
palak                                              1  1   22.566044
lady's finger                                      1  1   22.301918
brinjal                                            1  1   21.836514
banana                                             1  1   21.421793
potato                                             1  1   21.125746
radish                                             1  1   20.424635
gourd, pumpkin                                     1  1   20.167833
cabbage                                            1  1   19.741116
onion                                              1  1   17.685331
tomato                                             1  1   16.564789
sugar - P.D.S.                                     1  1   13.965524
kerosene-pds                                       1  1   13.944370
coconut: green                                     1  1    9.937186
salt                                               1  1    9.572224
wheat/atta - P.D.S.                                1  1    9.341238
rice - P.D.S.                                      1  1    8.995811
orange,mausami                                     1  1    7.427675
coconut                                            1  1    6.823189
candle                                             1  1    4.615317
firewood & chips                                   1  1    3.071869
matches                                            1  1    1.000000
papad, bhujia, namkeen, mixture, chanachur         1  1    1.000000
electricity                                        1  1    1.000000
cigarettes                                         1  1    1.000000
pan : leaf                                         1  1    1.000000
other vegetables                                   1  1    1.000000
```

Now compute expenditures on different items. The object `r` already knows what the estimated parameters are, and uses those automatically:

In [13]:
```
c=r.demands(median_x,p)
c
```

```
/opt/conda/lib/python3.9/site-packages/demands/_utils.py:52: UserWarning: Setting negati
ve values of beta to zero.
  warnings.warn('Setting negative values of beta to zero.')
```

Out[13]:
```
i
apple                         1.157931
arhar (tur)                   1.065204
banana                        1.499572
besan                         0.823195
black pepper                  0.581756
                                ...
tomato                        1.304990
turmeric                      0.605262
urd                           1.016945
wheat/atta - P.D.S.           1.131145
wheat/atta - other sources    1.395328
Name: quantities, Length: 90, dtype: float64
```

Now we can trace out demands for a household with median budget but varying prices of one good while holding other prices fixed:

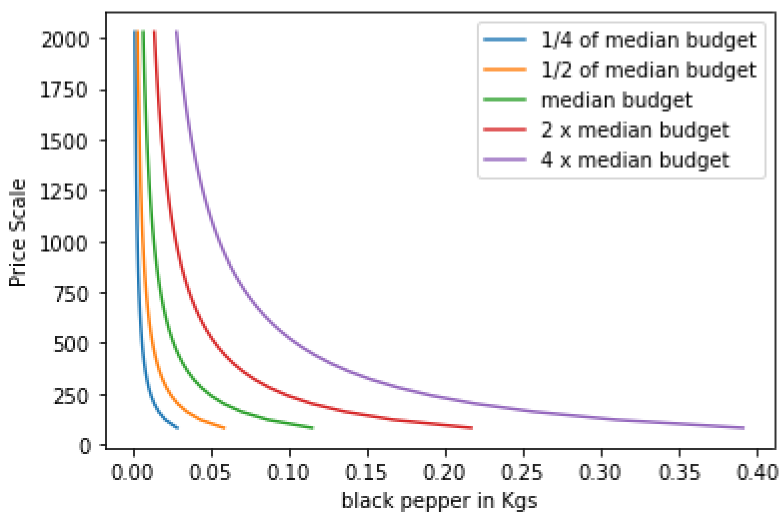
The `graph_demand` function takes in a food name and generate the demand curves for this good; each curve represent the demand for household of varying budget level with respect to the median budget.

Input Parameters:

- **food**: a string (any food name from the xhat df columns)

In [16]:
```
def graph_demand(product):
# Values for prices
    ref_price = r.prices.sel(i=product,t=t,m=m)
    P = np.linspace(ref_price/5,ref_price*5,50)

    def my_prices(p0,p=p,i=product):
        p = p.copy()
        p.loc[i] = p0*p.sel(i=i)
        return p

    for myx in [median_x*s for s in [.25,.5,1.,2,4]]:
        with warnings.catch_warnings():
            warnings.filterwarnings('ignore')
            plt.plot([r.demands(myx,my_prices(p0))[product] for p0 in P],P)
    plt.legend(['1/4 of median budget', '1/2 of median budget','median budget',
              '2 x median budget', '4 x median budget'])

    plt.xlabel("%s in Kgs" % product)
    plt.ylabel('Price Scale')
```

In [17]:
```
#example
#
graph_demand('black pepper')
```



In [18]:
```
#interactive presentation of demand for all products
#this step will take some time to run and respond
all_good = p_df.sort_values(by = 'prices', ascending=False).index

interact(graph_demand, product = all_good)
```

```
interactive(children=(Dropdown(description='product', options=('foreign liquor or refine
d liquor', 'coffee: po…
```

Out[18]:
```
<function __main__.graph_demand(product)>
```

The `graph_engel` function takes in a food name and generate an Engel's Law graph to demonstrate the relationship between total food expenditure and expenditure on a sigle food

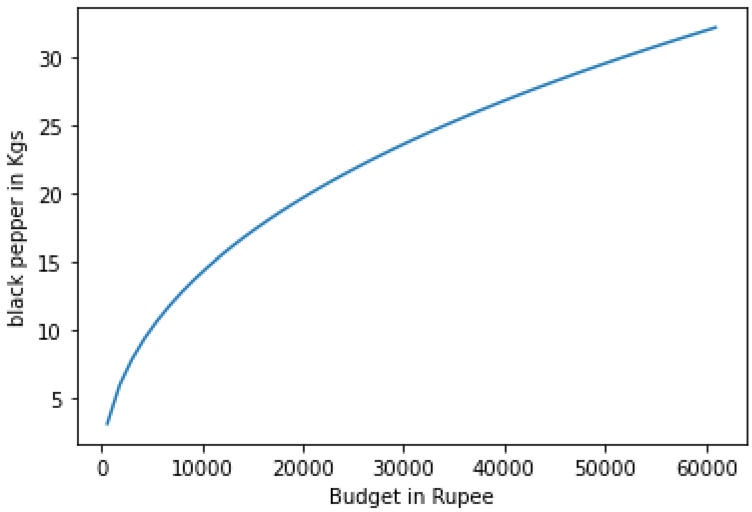
Input Parameters:

- **food**: a string (any food name from the xhat df columns)

In [19]:
```
def graph_engel(product):
    # Values for prices
    ref_price = r.prices.sel(i=product,t=t,m=m)

    # Range of budgets to consider
    X = np.linspace(median_x/10,median_x*10,50)

    plt.plot(X,[r.demands(x,ref_price)[product] for x in X])

    plt.ylabel("%s in Kgs" % product)
    plt.xlabel('Budget in Rupee')
```

In [20]:
```
#example
graph_engel('black pepper')
```

```
/opt/conda/lib/python3.9/site-packages/demands/_utils.py:52: UserWarning: Setting negati
ve values of beta to zero.
  warnings.warn('Setting negative values of beta to zero.')
```



In [27]:
```
#interactive presentation of engel curves for all products
# product is sorted based on their beta values (elasticity),
good_beta_sort = r.get_beta(as_df=True).dropna().sort_values(ascending=False).index
interact(graph_engel, product = good_beta_sort)
```

```
interactive(children=(Dropdown(description='product', options=('cashewnut', 'ghee', 'ele
ctricity ', 'carrot', …
```

Out[27]:
```
<function __main__.graph_engel(product)>
```

B. [A] Nutritional Adequacy

In [41]:
```
# Reference budget (find mean in reference period & market):
reference_x = r.get_predicted_expenditures().mean('j').sum('i').sel(t=t,m=m)

p = r.prices.sel(t=t,m=m,drop=True)
p = p.to_dataframe('i').squeeze().dropna()
p
```

Out[41]:
```
i
apple                        126.592908
arhar (tur)                   52.544116
banana                        21.421793
besan                         67.844577
black pepper                 405.984789
                                ...
```

```
tomato                         16.564789
turmeric                      188.681783
urd                            53.186266
wheat/atta - P.D.S.             9.341238
wheat/atta - other sources     37.774307
Name: i, Length: 84, dtype: float64
```

Nutritional Needs of Households

Our data on demand and nutrients is at the *household* level; we can't directly compare household level nutrition with individual level requirements. What we **can** do is add up minimum individual requirements, and see whether household total exceed these. This isn't a guarantee that all individuals have adequate nutrition (since the way food is allocated in the household might be quite unequal, or unrelated to individual requirements), but it is *necessary* if all individuals are to have adequate nutrition.

For the average household in Tamil Nadu, the number of different kinds of people can be computed by averaging over households:

In [29]:
```python
# In first round, averaged over households

zbar = r.z.sel(t=r.firstround,drop=True).mean(['j','m'])[:-1].squeeze() # Leave out log

zbar = zbar.to_dataframe().squeeze()
#on average, there's 3.66 individuals in a household in Tamil Nadu
zbar.sum()
```

Out[29]:
```
3.6645103054009325
```

Now, the inner/dot/matrix product between `zbar` and the `rda` DataFrame of requirements will give us minimum requirements for the average household:

In [30]:
```python
DRIs = pd.read_csv('Dietary Requirements - diet_minimums.csv')
# Define *minimums*
diet_min = DRIs.set_index('Nutrition')
```

In [31]:
```python
new_df = pd.DataFrame(index = diet_min.index)
new_df['Males 0-1'] =  diet_min['C 1-3'].to_list()
new_df['Females 0-1'] = diet_min['C 1-3'].to_list()
new_df['Males 1-5'] =  (np.array(diet_min['C 1-3']) + np.array(diet_min['M 4-8'])) / 2
new_df['Females 1-5'] =  (np.array(diet_min['C 1-3']) + np.array(diet_min['F 4-8'])) / 2
new_df['Males 5-10'] =  (np.array(diet_min['M 4-8']) + np.array(diet_min['M 9-13'])) / 2
new_df['Females 5-10'] =  (np.array(diet_min['M 4-8']) + np.array(diet_min['M 9-13'])) /
new_df['Males 10-15'] =  (np.array(diet_min['M 9-13']) + np.array(diet_min['M 14-18']))
new_df['Females 10-15'] =  (np.array(diet_min['F 9-13']) + np.array(diet_min['F 14-18'])
new_df['Males 15-20'] =  np.array(diet_min['M 14-18'])
new_df['Females 15-20'] =  np.array(diet_min['F 14-18'])
new_df['Males 20-30'] =  np.array(diet_min['M 19-30'])
new_df['Females 20-30'] =  np.array(diet_min['F 19-30'])
new_df['Males 30-50'] =  np.array(diet_min['M 31-50'])
new_df['Females 30-50'] =  np.array(diet_min['F 31-50'])
new_df['Males 50-60'] =  np.array(diet_min['M 51+'])
new_df['Males 60-100'] =  np.array(diet_min['M 51+'])
new_df['Females 50-60'] =  np.array(diet_min['F 51+'])
new_df['Females 60-100'] =  np.array(diet_min['F 51+'])
rda = new_df
```

In [32]:
```python
#check if all age-sex range is label correctly in rda and zbar
rda.columns.difference(zbar.index)
```

Out[32]:
```
Index([], dtype='object')
```

In [33]:

```
# May need to tweak types or alignment to match RDA and zbar types:
rda0,zbar0=rda.align(zbar,axis=1)

# This matrix product gives minimum nutrient requirements for average
# household
hh_rda = rda0.replace('',0)@zbar0

# RDA is /daily/, but  demands in our data are /monthly/:
hh_rda = hh_rda*30
hh_rda
```

Out[33]:

```
Nutrition
Energy                              207173.762600
Protein                               5008.580563
Fiber, total dietary                  2900.432676
Folate, DFE                          40465.924477
Calcium, Ca                         117632.540996
Carbohydrate, by difference          14291.590191
Iron, Fe                              1227.330375
Magnesium, Mg                        36235.843238
Niacin                                1527.752369
Phosphorus, P                        86586.422446
Potassium, K                        500589.514066
Riboflavin                             120.797954
Thiamin                                116.438544
Vitamin A, RAE                       80747.028735
Vitamin B-12                           242.795547
Vitamin B-6                            137.669174
Vitamin C, total ascorbic acid        7821.551076
Vitamin E (alpha-tocopherol)          1513.142771
Vitamin K (phylloquinone)             9999.112382
Zinc, Zn                               971.865503
dtype: float64
```

Nutritional Adequacy of Food Demands

Food Conversion Table

As usual, we need data to convert foods to nutrients:

In [34]:

```
#read the csv file containing all fdc codes for TN goods
fdc_codes = pd.read_csv('proj_4_fdc_codes_tamilnadu.csv - Sheet1.csv').set_index('Item')
fdc_codes = fdc_codes.reset_index()
fdc_codes
```

Out[34]:

	Item	ID
0	apple	1102644
1	arhar (tur)	1977550
2	banana	1102653
3	besan	2091506
4	black pepper	170931
...	...	...
69	tea; leaf	1104262
70	tomato	1103276
71	turmeric	172231
72	urd	1898206

73	wheat/atta - other sources	522973

74 rows × 2 columns

In [35]:

```
import fooddatacentral as fdc

apikey = 'CDXgPa1HVqJab8EFllem1ikOF75m2ELYwziKtICr'
D = {}
count = 0
for food in fdc_codes.Item.tolist():
    try:
        FDC = fdc_codes.loc[fdc_codes.Item==food,:].ID[count]
        count+=1
        print(FDC)
        D[food] = fdc.nutrients(apikey,FDC).Quantity
    except AttributeError:
        warnings.warn("Couldn't find FDC Code %s for food %s." % (food,FDC))

D = pd.DataFrame(D,dtype=float).fillna(0)

D
```

```
1102644
1977550
1102653
2091506
170931
1100621
2024758
1103343
1103193
1100517
1103345
2029648
170497
1648089
1100523
1100522
1104259
1919204
1155520
1102631
170922
168570
748278
577532
1028841
171907
2216557
1103354
1103844
1937534
175304
168448
2166704
1988217
1955347
1102665
1100536
1750348
1102666
1942595
1915741
2008520
1102594
```

```
2091229
1102670
1909132
1100404
598232
1103364
1102597
1889171
1103153
170917
168106
168414
170419
1103686
1102879
1103374
2057457
1102640
2129576
2077766
173468
1100464
1103933
1126152
1102697
1104274
1104262
1103276
172231
1898206
522973
```

Out[35]:

	apple	arhar (tur)	banana	besan	black pepper	bread (bakery)	brinjal	cabbage	carrot	cashewnut	...	sev nooc
Alanine	0.00	0.0	0.00	0.0	0.616	0.00	0.0	0.00	0.00	0.00	...	0
Alcohol, ethyl	0.00	0.0	0.00	0.0	0.000	0.00	0.0	0.00	0.00	0.00	...	0
Amino acids	0.00	0.0	0.00	0.0	0.000	0.00	0.0	0.00	0.00	0.00	...	0
Arginine	0.00	0.0	0.00	0.0	0.308	0.00	0.0	0.00	0.00	0.00	...	0
Ash	0.00	0.0	0.00	0.0	4.490	0.00	0.0	0.00	0.00	0.00	...	0
...	...	...	...	...	...	...	...	...	...	...	...	
Vitamin K (Menaquinone-4)	0.00	0.0	0.00	0.0	0.000	0.00	0.0	0.00	0.00	0.00	...	0
Vitamin K (phylloquinone)	2.20	0.0	0.50	0.0	163.700	0.20	0.0	38.20	13.20	36.80	...	1
Vitamins and Other Components	0.00	0.0	0.00	0.0	0.000	0.00	0.0	0.00	0.00	0.00	...	0
Water	85.56	0.0	74.91	0.0	12.460	35.70	0.0	90.39	88.29	1.64	...	66
Zinc, Zn	0.04	0.0	0.15	0.0	1.190	0.88	0.0	0.22	0.24	5.38	...	1

182 rows × 74 columns

In [36]:

```
#transpose and reformat
fct = D.T
```

Nutrient Demand

We can also use our demand functions to compute nutrition as a *function* of prices and budget.

In [39]:
```
import warnings

def my_prices(p0,p=p,i='apple'):
    """
    Set price of good i to p0, holding remaining prices fixed at values in p.
    """
    p = p.copy()
    p.loc[i] = p0
    return p.squeeze()

# x is income, p is a vector of prices
def nutrient_demand(x,p):
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        c = r.demands(x,p)

    fct0,c0 = fct.align(c,axis=0,join='inner')
    N = fct0.T@c0

    N = N.loc[~N.index.duplicated()]

    return N
```

With this `nutrient_demand` function in hand, we can see how nutrient outcomes vary with budget, given prices:

The `nut_vs_budget` function takes in a list of nutrient and see how nutrient outcomes vary with budget

Input Parameters:

- **nutrient**: a list of string of nutrient names
- **budget**: a reference x; we assume the median by defalt

In [45]:
```
def nut_vs_budget(nutrient, budget):
    X = np.linspace(budget/5,budget*5,50)

    df = pd.concat({myx:np.log(nutrient_demand(myx,p))[nutrient] for myx in X},axis=1).T
    ax = df.plot()

    ax.set_title('Nutrient Outcome v.s. Change in Budget')
    ax.set_xlabel('log budget in Rupee')
    ax.set_ylabel('log nutrient')
```

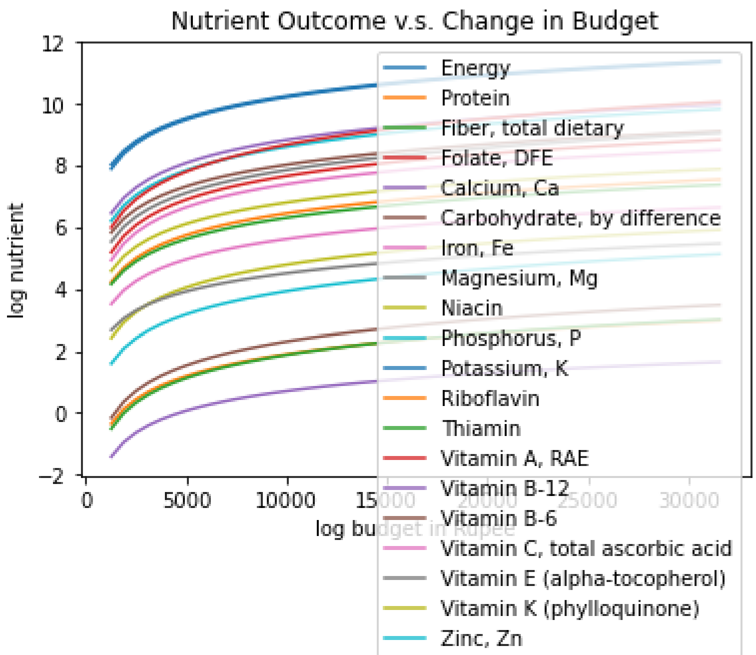
In [46]:
```
#example
#all nutrients, median budget as reference budget
AllNutrients = hh_rda.index.tolist()

nut_vs_budget(AllNutrients, budget = reference_x)
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
```

```
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
```

```
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
```

```
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



Now how does nutrition vary with prices?

The `nut_vs_prices` function takes in a list of nutrient and see how nutrient outcomes vary with changes in price for a specified food

Input Parameters:

- **nutrient**: a list of string of nutrient names
- **budget**: a reference x; we assume the median by defalt
- **good**: a specified food

In [47]:

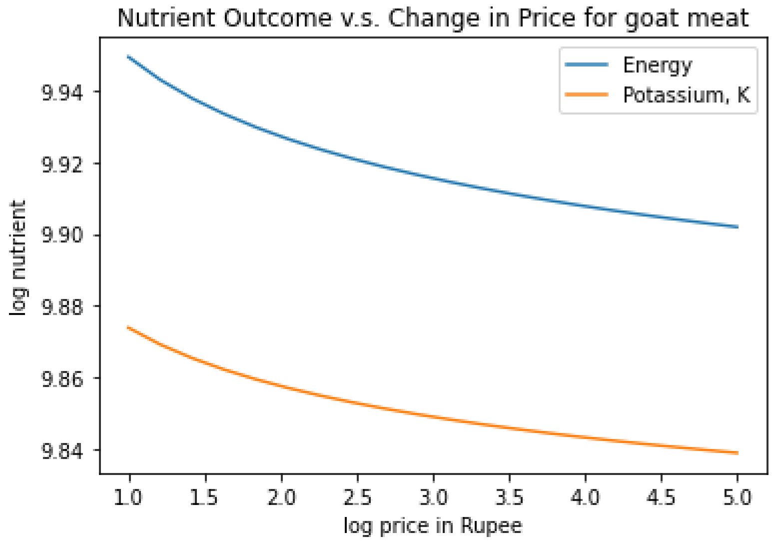
```
def nut_vs_prices(nutrient, budget, good):
    ref_price = r.prices.sel(i=good,t=t,m=m,drop=True)
    P = np.linspace(1,5,20).tolist()
    ndf = pd.DataFrame({p0:np.log(nutrient_demand(budget,my_prices(p0,i=good)))[nutrient

    ax = ndf.plot()

    ax.set_title(f"Nutrient Outcome v.s. Change in Price for {good}")
    ax.set_xlabel('log price in Rupee')
    ax.set_ylabel('log nutrient')
```

In [48]:

```
# example:
#goat meat; energy and potassium

KeyNutrients = ['Energy','Potassium, K']
nut_vs_prices(KeyNutrients, reference_x, 'goat meat')
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
```

```
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: div
ide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



Nutritional Adequacy

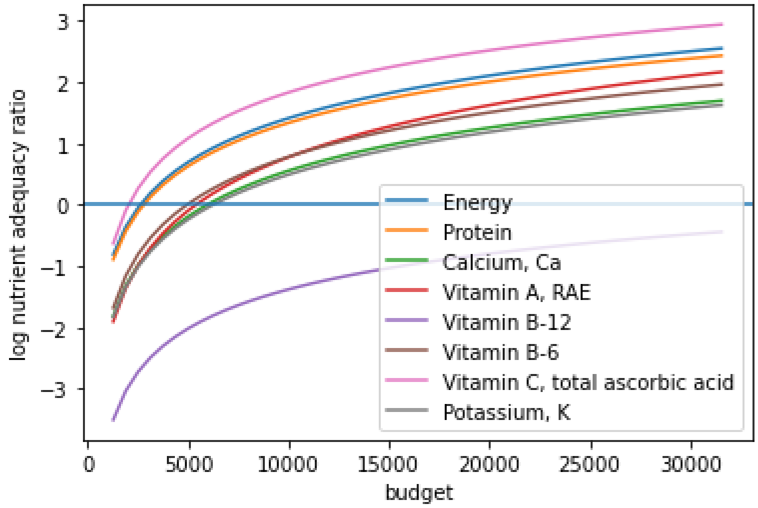
Since we can trace out demands for nutrients as a function of (x, p), and we've computed minimum nutritional requirements for the average household, we can *normalize* nutritional intake to check the adequacy of diet.

In [49]:
```
def nutrient_adequacy_ratio(x,p):
    return nutrient_demand(x,p)/(hh_rda/30)
```

In terms of normalized nutrients, any household with more than one unit of any given nutrient (or zero in logs) will be consuming a minimally adequate level of the nutrient; below this level there's clearly nutritional inadequacy. For this reason the ratio of actual nutrients to required nutrients is termed the "nutrient adequacy ratio," or NAR.

In [51]:
```
X = np.linspace(reference_x/5,reference_x*5,50)

UseNutrients = ['Energy',
                'Protein',
                'Calcium, Ca',
                'Vitamin A, RAE',
 'Vitamin B-12',
 'Vitamin B-6',
 'Vitamin C, total ascorbic acid',
'Potassium, K']

ndf = pd.concat({x:np.log(nutrient_adequacy_ratio(x,p))[UseNutrients] for x in X},axis=1

ax = ndf.plot()

ax.set_xlabel('budget')
ax.set_ylabel('log nutrient adequacy ratio')
ax.axhline(0)
```

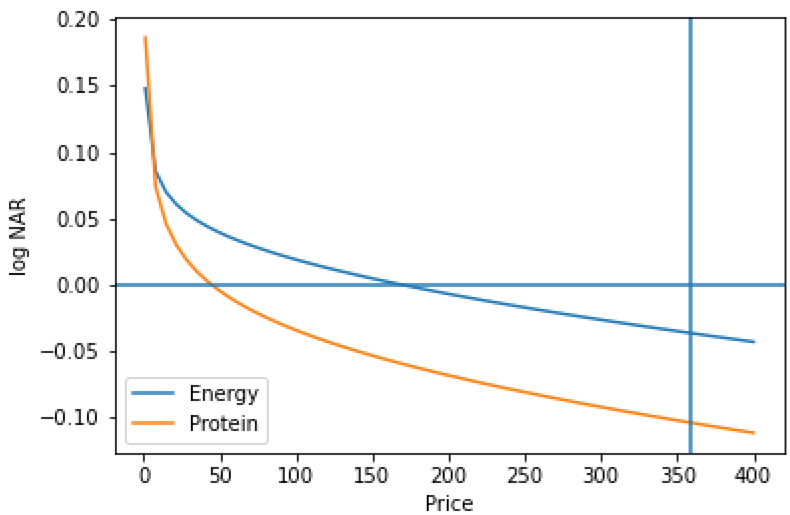
Out[51]:
```
<matplotlib.lines.Line2D at 0x7f3a7999fee0>
```



As before, we can also vary relative prices. Here we trace out nutritional adequacy varying the price of a single good:

In [55]:
```
poorer_x = reference_x/2.5

good = 'goat meat'

ExNutrients = ['Energy', 'Protein']

Pscale = np.linspace(1,400,60).tolist()

log_nar = {s0:np.log(nutrient_adequacy_ratio(poorer_x,my_prices(s0,p,i=good)))[ExNutrien

log_nar = pd.DataFrame(log_nar).T

ax = log_nar.plot(ylabel='log NAR',xlabel='Price')


ax.axhline(0)
ax.axvline(p[good])

#vertical line atural price of good
#horizaon line: if you are above, you have adequate nutrition
```

Out[55]: <matplotlib.lines.Line2D at 0x7f3a756995e0>



For West Bengal

We are going to replicate the code above for the second state that we are investigating; we have the code ready in the "draft" file in our github repo, but it is yet to be compiled

We are also finalizing our code for the policy portion

In [ ]:
```

```