Statistics
Assignment 2

1 Problem 3: Hypothesis testing

```
In [1]: import numpy as np
        import pandas as pd
        from math import sqrt
        import scipy.stats as stats
        import statsmodels.api as sm
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        import seaborn as sns
        sns.set_style("darkgrid")
        import warnings
        warnings.filterwarnings('ignore')
```

1.1 1

We start with the verication of the law of large numbers. Thus we check if an estimator converges (in probability) to its true value if the sample size increases.

1.1.1 1.a

Simulate samples of size $n = 100, \ldots, 100000$ from a normal distribution with mean 1 and variance 1, i.e. $N(1, 1)$. For each sample estimate the mean, the variance and store them.

```
In [138]: np.random.seed(0)

          lognmax = 5
          n_min = 10**2
          n_max = 10**lognmax
          ns = np.arange(n_min, n_max, 200)
          num_experiments = len(ns)
          print('{} samples in total. Some of them:'.format(num_experiments))
          mu1 = 1
          sigma1 = 1
          var1 = sigma1**2
          rv_norm1 = stats.norm(loc=mu1, scale=sigma1)
          sample_means = []
          sample_vars = []
```

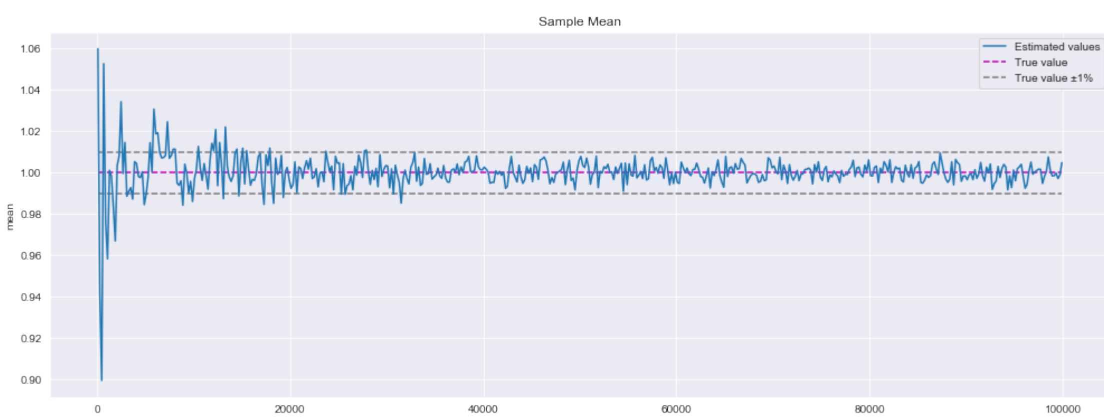
```
    priniter = 20
    for i, n in enumerate(ns):
        sample = rv_norm1.rvs(size=n)
        sample_mean = sample.mean()
        sample_var = sample.var(ddof=1)
        sample_means.append(sample_mean)
        sample_vars.append(sample_var)
        if i % priniter == 0:
            print('N({},{}) | Generated sample of size {:<{}}'\
                  .format(mu1, sigma1**2, n, lognmax+1) + \
              ' | Est. Mean: {:6.3f} | Est. Variance: {:6.3f}'.format(sample_mean,
                                                                       sample_var))
```

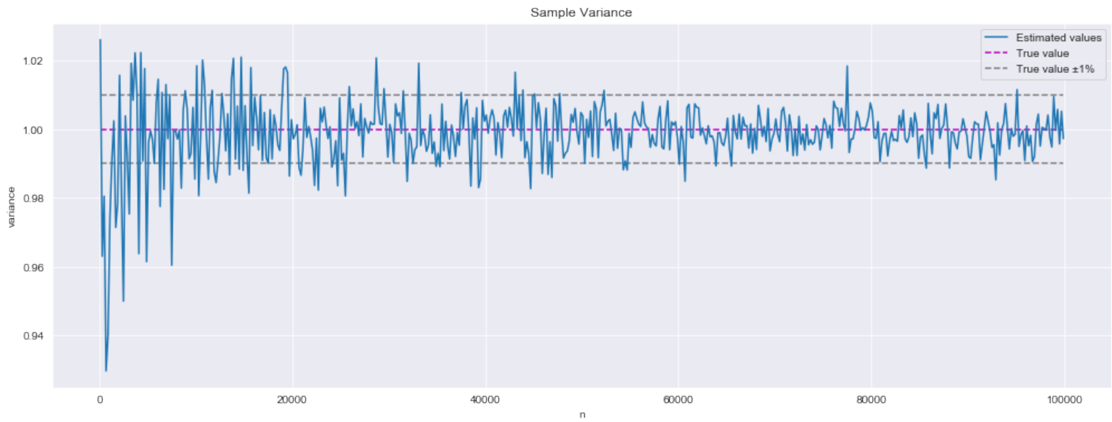
```
500 samples in total. Some of them:
N(1,1) | Generated sample of size 100    | Est. Mean:  1.060 | Est. Variance:  1.026
N(1,1) | Generated sample of size 4100   | Est. Mean:  1.004 | Est. Variance:  0.964
N(1,1) | Generated sample of size 8100   | Est. Mean:  1.011 | Est. Variance:  0.997
N(1,1) | Generated sample of size 12100  | Est. Mean:  1.011 | Est. Variance:  0.984
N(1,1) | Generated sample of size 16100  | Est. Mean:  0.997 | Est. Variance:  1.009
N(1,1) | Generated sample of size 20100  | Est. Mean:  0.992 | Est. Variance:  0.997
N(1,1) | Generated sample of size 24100  | Est. Mean:  1.000 | Est. Variance:  0.989
N(1,1) | Generated sample of size 28100  | Est. Mean:  1.001 | Est. Variance:  1.002
N(1,1) | Generated sample of size 32100  | Est. Mean:  0.997 | Est. Variance:  0.999
N(1,1) | Generated sample of size 36100  | Est. Mean:  0.997 | Est. Variance:  0.995
N(1,1) | Generated sample of size 40100  | Est. Mean:  1.003 | Est. Variance:  1.004
N(1,1) | Generated sample of size 44100  | Est. Mean:  0.995 | Est. Variance:  0.992
N(1,1) | Generated sample of size 48100  | Est. Mean:  1.001 | Est. Variance:  0.992
N(1,1) | Generated sample of size 52100  | Est. Mean:  1.001 | Est. Variance:  1.007
N(1,1) | Generated sample of size 56100  | Est. Mean:  0.999 | Est. Variance:  1.001
N(1,1) | Generated sample of size 60100  | Est. Mean:  0.995 | Est. Variance:  0.990
N(1,1) | Generated sample of size 64100  | Est. Mean:  1.001 | Est. Variance:  0.999
N(1,1) | Generated sample of size 68100  | Est. Mean:  0.999 | Est. Variance:  0.994
N(1,1) | Generated sample of size 72100  | Est. Mean:  0.996 | Est. Variance:  1.000
N(1,1) | Generated sample of size 76100  | Est. Mean:  0.998 | Est. Variance:  1.008
N(1,1) | Generated sample of size 80100  | Est. Mean:  0.998 | Est. Variance:  1.005
N(1,1) | Generated sample of size 84100  | Est. Mean:  0.998 | Est. Variance:  1.003
N(1,1) | Generated sample of size 88100  | Est. Mean:  0.995 | Est. Variance:  0.989
N(1,1) | Generated sample of size 92100  | Est. Mean:  1.003 | Est. Variance:  1.002
N(1,1) | Generated sample of size 96100  | Est. Mean:  0.992 | Est. Variance:  1.001
```

Plot the path of sample means and sample variances as function of n.

```
In [167]: delta = 0.01
          mpl.rcParams['figure.figsize'] = (17,6)

          def plot_parameter(ns, estimated_values, true_value,
                             delta=None,
```

```
                     cs=None, alpha=None,
                     parameter_name=''):
    '''
    delta -- constant small deviation from the true_value;
             the horizontal lines with values of
             `(1 + delta) * true_value` will be plotted
    cs -- deviations to construct the confidence intervals
    '''
    plt.title('Sample {}'.format(parameter_name))
    plt.xlabel('n')
    plt.ylabel(parameter_name.lower())
    plt.plot(ns, estimated_values, label='Estimated values')
    plt.hlines(true_value, ns[0], ns[-1], 'm', '--', label='True value')
    if delta is not None:
        plt.hlines((1+delta)*true_value, ns[0], ns[-1], 'gray', '--',
                   label='True value ś{}%'.format(int(delta*100)))
        plt.hlines((1-delta)*true_value, ns[0], ns[-1], 'gray', '--')
    if cs is not None:
        if isinstance(cs, tuple):
            cs_lw, cs_up = cs[0], cs[1]
        else:
            cs_lw, cs_up = estimated_values - cs, estimated_values + cs
        plt.plot(ns, cs_up, '--', color='gray',
                 label='{}% confidence interval for the {}'\
                  .format(int((1-alpha)*100), parameter_name.lower()))
        plt.plot(ns, cs_lw, '--', color='gray')
    plt.legend()
    plt.show()

plot_parameter(ns, sample_means, mu1, delta=delta, parameter_name='Mean')
plot_parameter(ns, sample_vars, var1, delta=delta, parameter_name='Variance')
```





The plots support the LLN. As n becomes bigger the estimated values become closer to the true values.

How many observations do we need in order to obtain an estimator which is close enough (ś1%) to the true value?

```
In [173]: def min_num_observations(ns, estimated_values, true_value, delta):
              index = [(np.array(estimated_values)[i:] > (1-delta)*true_value).all() and
                       (np.array(estimated_values)[i:] < (1+delta)*true_value).all()
                                            for i in range(len(ns))].index(True)
              return ns[index]

          min_observations_mean = min_num_observations(ns, sample_means, mu1, delta)
          min_observations_var = min_num_observations(ns, sample_vars, var1, delta)
          min_observations = max(min_observations_mean, min_observations_var)
          print('Experimentally we get that:')
          print('\tAfter {:5} observations mean estimator is close enough (ś1%) to the true mea
          print('\tAfter {:5} observations variance estimator is close enough (ś1%) to the true
          print('\t=> After {:5} observations estimations are close enough to the true mean and
```

```
Experimentally we get that:
        After 31700 observations mean estimator is close enough (ś1%) to the true mean
        After 95300 observations variance estimator is close enough (ś1%) to the true variance
        => After 95300 observations estimations are close enough to the true mean and variance
```

To get the number of observations theoretically we should use the known true value of σ and set the confidence level to $1-\alpha$ e.g. 0.95:

$$P\left(|\mu - \overline{X}| < z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}\right) > 1-\alpha \tag{1}$$

We want $|\mu - \overline{X}| < 0.01\mu = 0.01$

$$\Rightarrow z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}} = 0.01 \tag{2}$$

$$n = z^2_{1-\alpha/2}\frac{\sigma^2}{0.0001} \tag{3}$$

```
In [180]: alpha = 0.05
          n = stats.norm.ppf(1 - alpha/2) ** 2 * var1 / (0.01*mu1)**2
          print('We need more than {:.0f} observations to obtain an estimator close enough (ś1
```

```
We need more than 38415 observations to obtain an estimator close enough (ś1%) to the true valu
```

1.b Add to the plot the 95% condence intervals. These have to be constructed manually. Provide their interpretation.

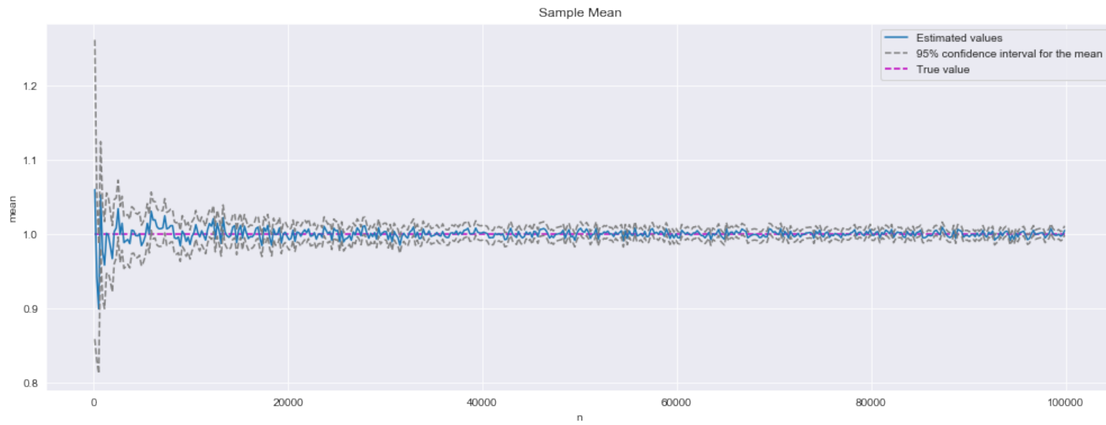
For the mean the confidence interval is provided with unknown σ (more often we don't now the true variance):

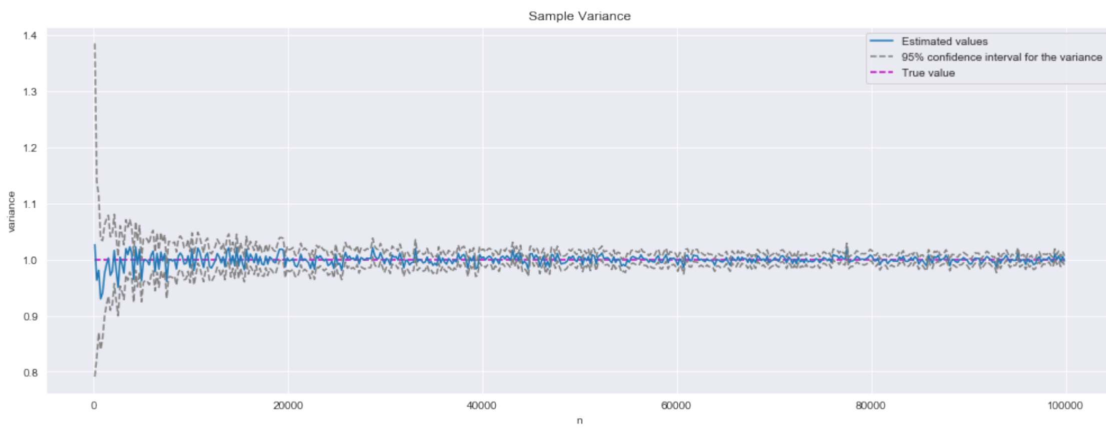
$$\left(\overline{X} - t_{n-1;1-\alpha/2}\frac{S}{\sqrt{n}}; \overline{X} + t_{n-1;1-\alpha/2}\frac{S}{\sqrt{n}}\right) \tag{4}$$

Analogously the confidence interval for the variance is constructed as we don't know the true mean:

$$\left(\frac{(n-1)S^2}{\chi^2_{n-1;1-\alpha/2}}; \frac{(n-1)S^2}{\chi^2_{n-1;\alpha/2}}\right) \tag{5}$$

```
In [172]: alpha = 0.05

          cs = np.array([stats.t.ppf(1-alpha/2, df=n-1) * np.sqrt(S2/n)
                         for (n,S2) in zip(ns, sample_vars)])
          plot_parameter(ns, sample_means, mu1, cs=cs, alpha=alpha,
                         parameter_name='Mean')

          cs_lw = np.array([S2*(n-1) / stats.chi2.ppf(1-alpha/2, df=n-1)
                            for (n,S2) in zip(ns, sample_vars)])
          cs_up = np.array([S2*(n-1) / stats.chi2.ppf(alpha/2, df=n-1)
                            for (n,S2) in zip(ns, sample_vars)])
          plot_parameter(ns, sample_vars, var1, cs=(cs_lw, cs_up),
                         alpha=alpha, parameter_name='Variance')
```





With 95% confidence we are sure that the true parameter value is inside the confidence interval.

1.2 2

The 2nd objective of this part is to get more feeling for the ML estimation procedures. The estimation for non-standard distributions/models usually follows the maximum likelihood principle. The t-distribution is a popular alternative if the sample distribution is symmetric but exhibits heavier tails compared to the normal distribution.

2 (a) Let $x_1, \ldots, x_n$ be a given sample. We assume that it stems from a t-distribution with an unknown number of degrees of freedom. Write down the corresponding likelihood function.

$$X_1, \ldots, X_n \sim t_d: \qquad f(x_i) = \frac{(1 + \frac{x_i^2}{d})^{-\frac{d+1}{2}}}{B(d/2, 1/2)\sqrt{d}} \tag{6}$$

$$L_d(x_1,\ldots,x_n) = \prod_{i=1}^{n} \frac{(1+\frac{x_i^2}{d})^{-\frac{d+1}{2}}}{B(d/2,1/2)\sqrt{d}} = \frac{\prod_{i=1}^{n}(1+\frac{x_i^2}{d})^{-\frac{d+1}{2}}}{B^n(d/2,1/2)\cdot d^{n/2}} \tag{7}$$

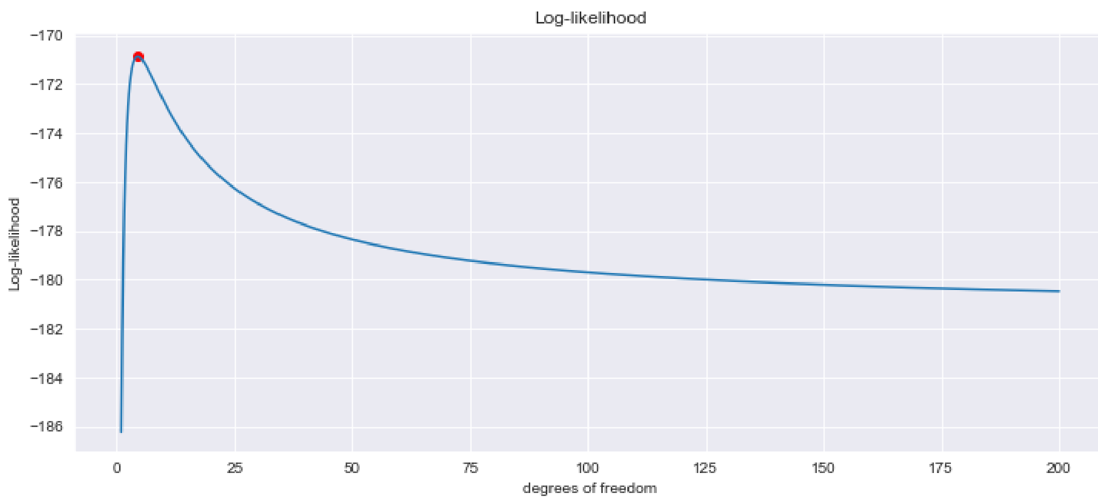
$$\ln L_d(x_1,\ldots,x_n) = \frac{-\frac{d+1}{2}\sum_{i=1}^{n}\ln(1+\frac{x_i^2}{d})}{n\ln(B(d/2,1/2)\cdot\sqrt{d})} \tag{8}$$

2 (b) Simulate a sample of size n = 100 from t_5 . Maximize the likelihood function (numerically) for the given sample and obtain the ML estimator of the number of degrees of freedom.

```
In [18]: def log_likelihood(pdfs):
             return np.sum(np.log(pdfs))
```

```
In [185]: mpl.rcParams['figure.figsize'] = (12, 5)
          np.random.seed(100)

          d_true = 5
          n = 100
          rv_st = stats.t(df=d_true)
          sample_st = rv_st.rvs(size=n)

          ds = np.linspace(1,200,1000)
          Ls = []
          for d in ds:
              pdfs = stats.t(df=d).pdf(sample_st)
          #     plt.scatter(sample_st, pdfs)
              Ls.append(log_likelihood(pdfs))
          # plt.show()
          d_est = ds[np.argmax(Ls)]
          print('df* = {:.2f} maximizes likelihood function'.format(d_est))

          plt.title('Log-likelihood')
          plt.xlabel('degrees of freedom')
          plt.ylabel('Log-likelihood')
          plt.plot(ds, Ls)
          plt.scatter(d_est, np.max(Ls), color='red')
          plt.show()
```

```
df* = 4.59 maximizes likelihood function
```



2 (c) The classical theoretical t-distribution has zero mean and variance df/(df 2). A real sample does not have exactly these moments. How would you proceed if you need to t a t-distribution to real data?

If we transform random variable X s.t. $Y = \sigma X + \mu$ then $F_Y(x) = F_X(\frac{x-\mu}{\sigma})$ and $f_Y(x) = F'_X(\frac{x-\mu}{\sigma}) = \frac{1}{\sigma} f_X(\frac{x-\mu}{\sigma})$
We can use it with transformed Student r.v. to calculate likekihood:

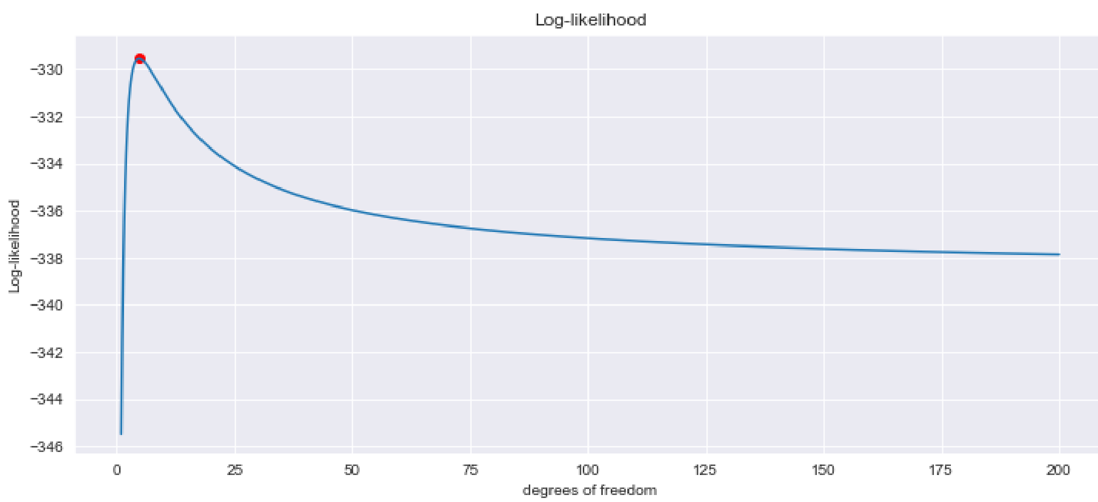
$$L(x_1, \dots, x_n) = \prod_{i=1}^{n} \frac{1}{\sigma} f_X\left(\frac{x-\mu}{\sigma}\right) \tag{9}$$

As was shown in previous assignment we can transform r. v. X from Student distribution with n degrees of freedom s.t. it has mean μ and variance σ^2: $Y = \sigma\sqrt{\frac{n-2}{n}}X + \mu$. So we can estimate n using ML and thus fit t_n-distribution to our data.

```
In [197]: mpl.rcParams['figure.figsize'] = (12, 5)
          np.random.seed(100)

          d_true = 5
          n = 100

          mu = 10
          sigma = 5
          sample_st = mu + sigma * np.sqrt((df-2) / df) * stats.t(df=d_true).rvs(size=n)

          ds = np.linspace(1,200,1000)
          Ls = []
          for d in ds:
              pdfs = stats.t(df=d).pdf((sample_st - mu) / sigma) / sigma
```

```
    Ls.append(log_likelihood(pdfs))
d_est = ds[np.argmax(Ls)]
print('df* = {:.2f} maximizes likelihood function'.format(d_est))

plt.title('Log-likelihood')
plt.xlabel('degrees of freedom')
plt.ylabel('Log-likelihood')
plt.plot(ds, Ls)
plt.scatter(d_est, np.max(Ls), color='red')
plt.show()
```

```
df* = 4.78 maximizes likelihood function
```



1.3 3

The 3rd objective is check if the probability of type 1 error (size of a test) is correctly attained by a simple two-sided test for the mean.

3 (a) Simulate a sample of length $n = 100$ from a normal distribution with mean $\mu_0 = 500$ and variance $\sigma_2 = 50$. (Note: you may use the transformation $X = \mu + \sigma Z$, where $Z \sim N(0, 1)$.) The objective is to test the null hypothesis $H_0 : \mu = 500$. Assume that σ_2 has to be estimated. Compute the test statistics using the formulas in the lecture; determine the rejection area for $\alpha = 0.04$ and decide if H_0 can to be rejected.

```
In [23]: np.random.seed(0)

         n = 100
         mu = 500
         sigma = 50
```

```
        rv = stats.norm(loc=mu, scale=sigma)
        x = rv.rvs(size=n)
```

```
In [24]: def normal_cdf(x):
             return stats.norm().cdf(x)

         def normal_ppf(F):
             return stats.norm().ppf(F)

         def student_cdf(df, x):
             return stats.t(df=df).cdf(x)

         def student_ppf(df, F):
             return stats.t(df=df).ppf(F)

         def two_sided_Z_test(x, a, alpha=None, sigma=None, out=True):
             '''
             H0: mu == a
             H1: mu != a
             alpha -- significance level
             '''
             if out:
                 print('\nTwo-sided Z-test {}'.format('(sigma is unknown)'
                                                      if sigma is None else ''))
                 print('H0: mu == {}'.format(a))
                 print('H1: mu != {}'.format(a))
             n = len(x)
             sample_mean = np.mean(x)
         #     print('\tSample mean: {:.2f}'.format(sample_mean))
             if sigma is None:
                 sample_var = np.var(x, ddof=1)
         #         print('\tSample variance: {:.2f}'.format(sample_var))
                 v = (sample_mean - a) * sqrt(n) / sqrt(sample_var)
                 p_value = 2 * student_cdf(n-1, v) if v < 0 else \
                           2 * (1 - student_cdf(n-1, v))
                 if alpha is not None:
                     t_crit = student_ppf(n-1, 1-alpha/2)
             else:
                 v = (sample_mean - a) * sqrt(n) / sigma
                 p_value = 2 * normal_cdf(v) if v < 0 else 2 * (1 - normal_cdf(v))
                 if alpha is not None:
                     t_crit = normal_ppf(1-alpha/2)

             if alpha is not None:
                 rejected = v < -t_crit or v > t_crit
                 if out:
                     print('The rejection area is: (-inf, -{:.3f})  ({:.3f}, inf)'\
```

```
                  .format(t_crit, t_crit))
            print('t_stat = {:.3f} is {} in the rejection area.'\
                  .format(v, '' if rejected else 'not'))
            if not rejected:
                print('With significance level of {}% we cannot reject H0.'\
                      .format(alpha*100, a))
            else:
                print('With significance level of {}% we can reject H0.')
        return p_value, rejected
    if out:
        print('p-value is {:.3f}'.format(p_value))
        print('For all  > {:.3f} we can reject H0.'.format(p_value))
    return p_value

_, _ = two_sided_Z_test(x, mu, 0.04)
```

```
Two-sided Z-test (sigma is unknown)
H0: mu == 500
H1: mu != 500
The rejection area is: (-inf, -2.081)  (2.081, inf)
t_stat = 0.590 is not in the rejection area.
With significance level of 4.0% we cannot reject H0.
```

3 (b) Determine the p-values using the formulas from the lecture and compare/check the results using a build-in function for this test in R or Python.

```
In [31]: p_value = two_sided_Z_test(x, mu)

         tstat, pvalue = stats.ttest_1samp(x, mu)
         print('\np-value from `scipy.stats` is {:.3f}'.format(pvalue))
```

```
Two-sided Z-test (sigma is unknown)
H0: mu == 500
H1: mu != 500
p-value is 0.556
For all  > 0.556 we can reject H0.

p-value from `scipy.stats` is 0.556
```

So implemented p-value is the same as within `scipy.stats`.

3 (c) Simulate $M = 1000$ samples of size $n = 100$ and with $_0 = 500$ and variance $\sigma^2 = 50$. For each sample i run the test (using a standard function) and set $p_i = 0$ if H_0 is not rejected and $p_i = 1$ if rejected. Compute $\hat{\alpha} = \frac{1}{M} \sum_{i=1}^{M} p_i$. $\hat{\alpha}$ is the empirical condence level (empirical size) of the test. Compare $\hat{\alpha}$ to α? Do you expect the dierence to be large or small and why? Relate it to the assumptions of the test.

```
In [111]: M = 1000
          n = 100
          mu = 500
          sigma = np.sqrt(50)
          alpha = 0.04

          rv = stats.norm(loc=mu, scale=sigma)
          X = rv.rvs(size=n*M).reshape((M,n))
          P = []
          for x in X:
              _, pvalue = sm.stats.ztest(x, value=mu)
              rejected = alpha > pvalue
              P.append(int(rejected))
```

```
In [112]: print('Empirical condence level: {}, level of significance  {}'\
                .format(np.mean(P), alpha))
```

```
Empirical condence level: 0.044, level of significance  0.04
```

The expectations are that the empirical confidence level would be close to α. Since we assume that data is normally distributed we should get the type 1 error with probability α. So for our 1000 experiments we should get roughly $1000\alpha = 40$ rejections, indeed we got 44. The bigger is M the closer $\hat{\alpha}$ should be to α, due to the Law of Large Numbers.

3 (d) Assume now that one of the assumptions is not satised. For example, the data is in fact not normal. Repeat the above analysis, but simulate the data from the t-distribution with 3 degrees of freedom. Give motivation and justication for the new values of $\hat{\alpha}$?

```
In [123]: M = 1000
          n = 100
          df = 3

          rv = stats.t(df=df)
          X = rv.rvs(size=n*M).reshape((M,n))
          X = mu + sigma * np.sqrt((df-2) / df) * X
          P = []
          for x in X:
              _, pvalue = sm.stats.ztest(x, value=mu)
              rejected = alpha > pvalue
              P.append(int(rejected))
```

```
In [124]: print('Empirical condence level: {}'.format(np.mean(P), alpha))
```

```
Empirical condence level: 0.033
```

We should also get empirical value relatively close to the level of significance since the distribution of the data is close to normal. Nevertheless the assumptions on the normal distribution are not fullfiled so we may need a bigger number of experiments to get a very close estimate.

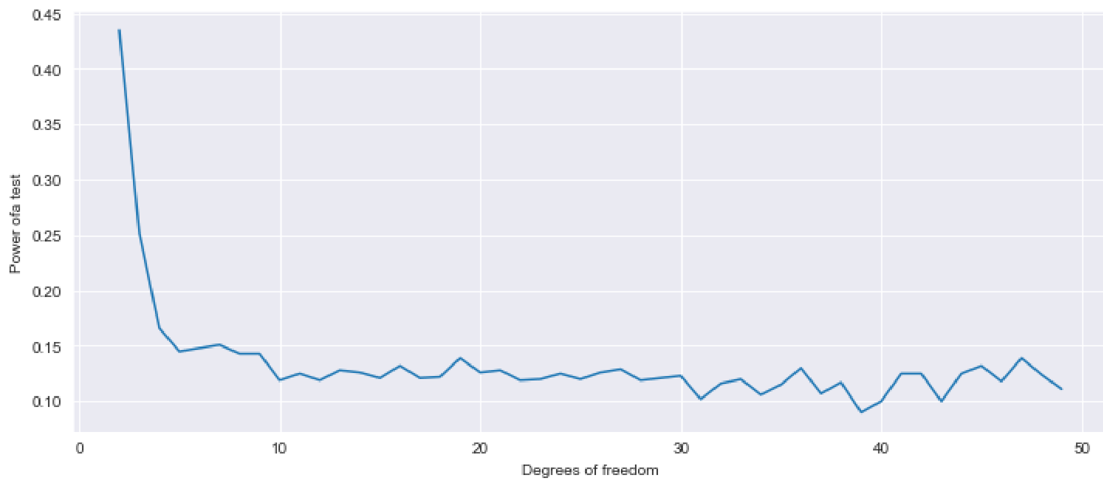
3 (e) Power of a test: The rst objective is to assess the probability of type 2 error (power of a test) of goodness-of-t test. Goodness-of-t tests for the normal distribution are of key importance in statistics, since they allow to verify the distributional assumptions required in many models. Here we check the power of the Kolmogorov-Smirnov test, i.e. is the test capable to detect deviations from normality?

Simulate $M = 1000$ samples of size 100 from a t-distribution with $df = 2, \ldots, 50$ degrees of freedom. For each sample run the Kolmogorov-Smirnov test and count the cases when the H_0 of normality is correctly rejected (for each df). How would you use this quantity to estimate the power of the test? Make an appropriate plot with the df on the X-axis. (Note: the t-distribution converges to the normal distribution as df tends to innity. For $df > 50$ the distributions are almost identical. Discuss the plot and draw conclusions about the reliability of the test.

```
In [131]: np.random.seed(0)

          M = 1000
          n = 100
          dfs = np.arange(2,50)
          print('H0: X ~ Normal')
          print('H1: X !~ Normal')
          alphas = []
          for df in dfs:
              rv = stats.t(df=df)
              X = rv.rvs(size=n*M).reshape((M,n))
              P = []
              for x in X:
                  ksstat, pvalue = stats.kstest(x, 'norm')
                  rejected = ksstat > pvalue
                  P.append(int(rejected))
              alphas.append(np.mean(P))
```

```
H0: X ~ Normal
H1: X !~ Normal
```

```
In [135]: plt.plot(dfs, alphas)
          plt.ylabel('Power ofa test')
          plt.xlabel('Degrees of freedom')
          plt.show()
```



The data is not distributed normally so H_0 of the Kolmogorov-Smirnov test should be rejected. A test's power $(1-\beta)$ is the probability that the test rejects the H_0 when H_1 is true. So the frequency of rejection H_0 with our data can be used as a direct estimation of a test's power. From the plot we see that as df grows the H_0 is rejected less frequently, meaning that the test's power decreases. This is expectable since the t-distribution becomes closer to the normal distribution with increasing df.

Problem4

1 Problem 4: Linear regression analysis

A telephone service provider aims to decrease the churn rate and analyses the data and service usage of 1000 clients. The following variables are used in the study

`tenure` - month a client;
`age` - age in years;
`marital status` - marital status (1 - married, 0 - single);
`address` - years at the current address;
`income` - household income in Euro;
`ed` - education (5 categories: Did not complete high school; High school degree; Some college; College degree; Post-undergraduate degree);
`retire` - retired (0 - no, 1 - yes);
`gender` - gender (0 - male, 1 - female);
`longmon` - long distance calls last month;
`wiremon` - internet use last month;
`churn` - 1 if the contract was terminated last month and 0 else

The overall objective is to analyze the service usage using `longmon` as the dependent variables and the remaining variables as explanatory.

In [489]:
```
import numpy as np
import pandas as pd
import scipy as sp
import scipy.stats as stats
import statsmodels.api as sm
from sklearn import linear_model, metrics
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
sns.set_style("darkgrid")
import warnings
warnings.filterwarnings('ignore')
from utils import *

df_raw = pd.read_csv('data/telco.txt', sep='\t')
N = len(df_raw)
df_raw.head()
```

```
Out[489]:    tenure  age    marital  address  income                         ed  \
          1      13   44    Married        9      64             College degree
          2      11   33    Married        7     136     Post-undergraduate degree
          3      68   52    Married       24     116  Did not complete high school
          4      33   33  Unmarried       12      33            High school degree
          5      23   30    Married        9      30  Did not complete high school

             employ retire  gender  longmon  wiremon churn
          1       5     No    Male     3.70      0.0   Yes
          2       5     No    Male     4.40     35.7   Yes
          3      29     No  Female    18.15      0.0    No
          4       0     No  Female     9.45      0.0   Yes
          5       2     No    Male     6.30      0.0    No
```

1.1 1.

Have a closer look at the denitions of the variables and analyze which of them might require a separate treatment. Consider for example the variable `ed`. There are two possibilities how the variable `ed` can be included into the model (one with dummy variables, the other one without dummies). Think about these two approaches and suggest which approach is more appropriate. Motivate your decision.

```
In [2]: print('Data types:\n{}'.format(df_raw.dtypes))
```

```
Data types:
tenure       int64
age          int64
marital     object
address      int64
income       int64
ed          object
employ       int64
retire      object
gender      object
longmon    float64
wiremon    float64
churn       object
dtype: object
```

```
In [3]: # Group variables by the type of the scale
        intervals = list(df_raw.dtypes[(df_raw.dtypes == np.int64) |
                                       (df_raw.dtypes == np.float64)].index)
                  # ['tenure', 'age', 'address', 'income',
                  #  'employ', 'longmon', 'wiremon']

        ordinals = ['ed']

        nominals = list(set(df_raw.dtypes.index).difference(intervals, ordinals))
```

```
            # ['marital', 'retire', 'gender', 'churn']

df_intervals = df_raw[intervals]
df_ordinals = df_raw[ordinals]
df_nominals = df_raw[nominals]
```

In [4]:
```
# Look at interval scaled variables
stats_descr = sp.stats.describe(df_intervals)
describe_intervals = df_intervals.describe(percentiles=[.1, .25, .5, .75, .9]).append(
    [pd.Series(stats_descr.skewness, index=intervals, name='skew'),
     pd.Series(stats_descr.variance, index=intervals, name='var')])
describe_intervals = describe_intervals.drop('count',0)
describe_intervals
```

Out[4]:
```
          tenure         age     address        income      employ  \
mean   35.526000   41.684000   11.551000     77.535000   10.987000
std    21.359812   12.558816   10.086681    107.044165   10.082087
min     1.000000   18.000000    0.000000      9.000000    0.000000
10%     7.000000   26.000000    1.000000     21.000000    0.000000
25%    17.000000   32.000000    3.000000     29.000000    3.000000
50%    34.000000   40.000000    9.000000     47.000000    8.000000
75%    54.000000   51.000000   18.000000     83.000000   17.000000
90%    66.000000   59.000000   26.100000    155.400000   25.000000
max    72.000000   77.000000   55.000000   1668.000000   47.000000
skew    0.111692    0.356128    1.104586      6.633303    1.059457
var   456.241566  157.723868  101.741140  11458.453228  101.648479

         longmon     wiremon
mean   11.723100   11.583900
std    10.363486   19.719426
min     0.900000    0.000000
10%     3.645000    0.000000
25%     5.200000    0.000000
50%     8.525000    0.000000
75%    14.412500   24.712500
90%    23.960000   42.110000
max    99.950000  111.950000
skew    2.961653    1.601274
var   107.401848  388.855747
```

In [5]:
```
# Look at nominal scaled variables
df_nominals.describe()
```

Out[5]:
```
          marital churn retire  gender
count        1000  1000   1000    1000
unique          2     2      2       2
top     Unmarried    No     No  Female
freq          505   726    953     517
```

```
In [6]: # Look at ordinal scaled variables
        df_ordinals.describe()
```

```
Out[6]:                       ed
        count               1000
        unique                 5
        top      High school degree
        freq                 287
```

The variable ed can be represented as dummy variables:

```
In [435]: ed_dummies = pd.get_dummies(df_raw.ed)
          ed_names_dict = {'College degree': 'ed_college',
                           'Did not complete high school': 'ed_no',
                           'High school degree': 'ed_highschool',
                           'Post-undergraduate degree': 'ed_postgr',
                           'Some college': 'ed_somecollege'}
          ed_dummies = ed_dummies.rename(columns=ed_names_dict).drop(['ed_no'], 1)
          ed_dummies.head()
```

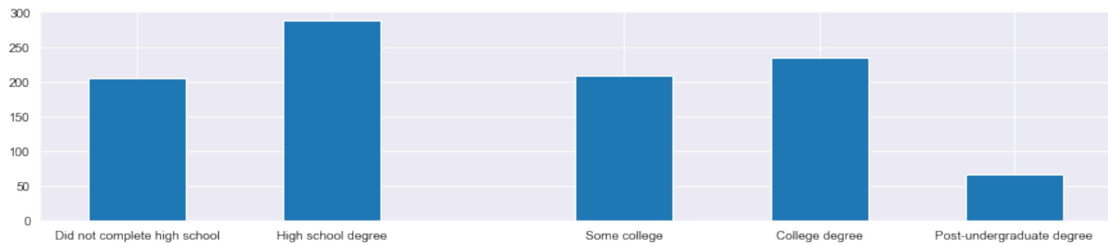
```
Out[435]:    ed_college  ed_highschool  ed_postgr  ed_somecollege
          1           1              0          0               0
          2           0              0          1               0
          3           0              0          0               0
          4           0              1          0               0
          5           0              0          0               0
```

```
In [8]: df_dummies = pd.concat((df_raw.drop('ed', 1), ed_dummies), 1)
        df_dummies['marital'] = df_dummies.marital.map(dict(Married=1, Unmarried=0))
        df_dummies['retire'] = df_dummies.retire.map(dict(Yes=1, No=0))
        df_dummies['gender'] = df_dummies.gender.map(dict(Female=1, Male=0))
        df_dummies['churn'] = df_dummies.churn.map(dict(Yes=1, No=0))
        df_dummies.head()
```

```
Out[8]:    tenure  age  marital  address  income  employ  retire  gender  longmon  \
        1      13   44        1        9      64       5       0       0     3.70
        2      11   33        1        7     136       5       0       0     4.40
        3      68   52        1       24     116      29       0       1    18.15
        4      33   33        0       12      33       0       0       1     9.45
        5      23   30        1        9      30       2       0       0     6.30

           wiremon  churn  ed_college  ed_highschool  ed_postgr  ed_somecollege
        1      0.0      1           1              0          0               0
        2     35.7      1           0              0          1               0
        3      0.0      0           0              0          0               0
        4      0.0      1           0              1          0               0
        5      0.0      0           0              0          0               0
```

Another option is to use the property of the order scaled variable and map the values to their ranks:

```
In [457]: ed_order_dict = {'Did not complete high school': 0,
                           'High school degree': 1,
                           'Some college': 2,
                           'College degree': 3,
                           'Post-undergraduate degree': 4}
          df_ordered = df_raw.copy()
          df_ordered['ed'] = df_ordered.ed.map(ed_order_dict)
          mpl.rcParams['figure.figsize'] = (15, 3)
          plt.hist(df_ordered.ed, align='mid')
          plt.xticks([0.2,1,2.2,3,3.8], ed_order_dict.keys())
          plt.show()
          df_ordered.head()
```




```
Out[457]:    tenure  age    marital  address  income  ed  employ retire  gender  \
          1      13   44    Married        9      64   3       5     No    Male
          2      11   33    Married        7     136   4       5     No    Male
          3      68   52    Married       24     116   0      29     No  Female
          4      33   33  Unmarried       12      33   1       0     No  Female
          5      23   30    Married        9      30   0       2     No    Male

             longmon  wiremon churn
          1     3.70      0.0   Yes
          2     4.40     35.7   Yes
          3    18.15      0.0    No
          4     9.45      0.0   Yes
          5     6.30      0.0    No
```

On the one hand, the second approach should be more appropriate since we include the information about the order of `ed` e.g. some education is better or higher than no education at all.

On the other hand with this order we "fix" the difference between degrees and say e.g. that `Post-undergraduate degree` is higher than `College degree` as much as `Some college` is higher than `High school degree` etc. while this may not be true. But with dummy variables this is not fixed, so it is suggested to use futher.

```
In [10]: df = df_dummies.copy()
```

1.2 2.

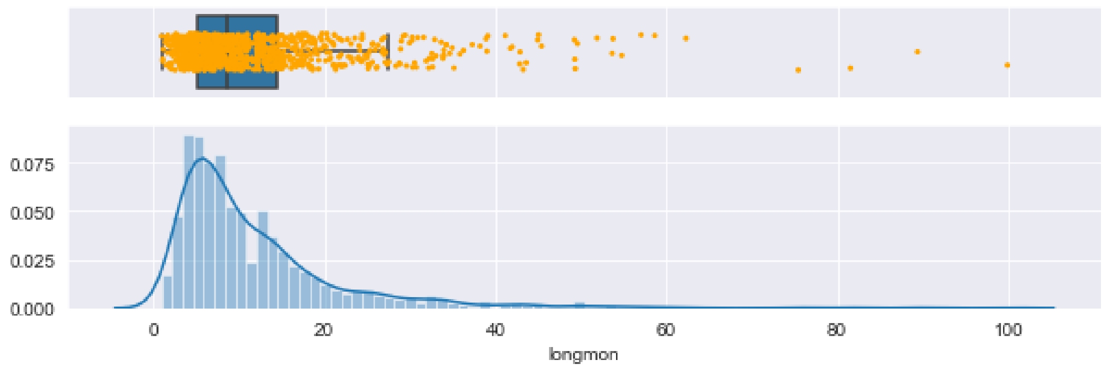
Consider now the dependent variable `longmon` and the interval (metric) scaled explanatory variables. Plot these data and decide if you wish to transform these x-variables and if there is a need to transform the y variable. You can also use some measure of skewness to decide about y. The variable `wiremon` shows a very specic pattern. How would you take it into account?

```
In [488]: mpl.rcParams['figure.figsize'] = (17,5)
```

`longmon` (long distance calls last month)

```
In [12]: column = 'longmon'
         represent_distribution(df[column], varname=column)
         print(describe_intervals.longmon)
         skewness = sp.stats.skew(df[column])
         print('The distribution is {}.'.format('right-skewed' if skewness > 0 else 'left-skewe
```

```
Variable `longmon`
Number of observations: 1000
```

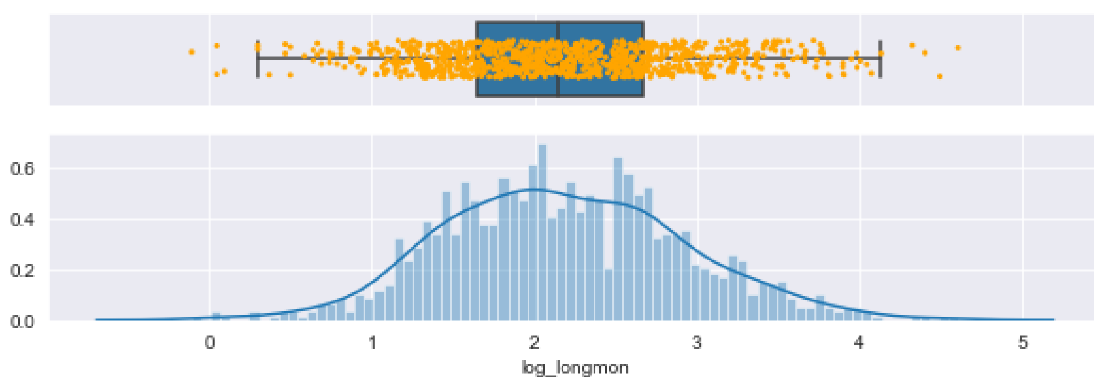



```
mean      11.723100
std       10.363486
min        0.900000
10%        3.645000
25%        5.200000
50%        8.525000
75%       14.412500
90%       23.960000
max       99.950000
skew       2.961653
var      107.401848
Name: longmon, dtype: float64
The distribution is right-skewed.
```

We transform this variable `longmon` into `log(longmon)`:

```
In [13]: df['log_{}'.format(column)] = np.log(df[column])
         column = 'log_{}'.format(column)
         represent_distribution(df[column], varname=column)
         # stats_descr = sp.stats.describe(df[['log_longmon']])
         # df[['log_longmon']].describe(percentiles=[.1, .25, .5, .75, .9]).append(
         # [pd.Series(stats_descr.skewness, index=['log_longmon'], name='skew'),
         # pd.Series(stats_descr.variance, index=['log_longmon'], name='var')]).drop('count',0)
```

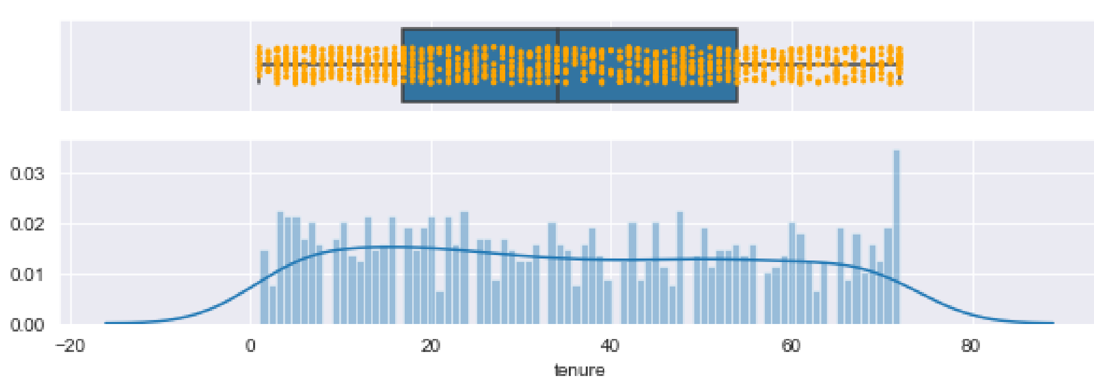
```
Variable `log_longmon`
Number of observations: 1000
```



tenure

```
In [14]: column = 'tenure'
         represent_distribution(df[column], varname=column)
         print(describe_intervals[column])
```

```
Variable `tenure`
Number of observations: 1000
```

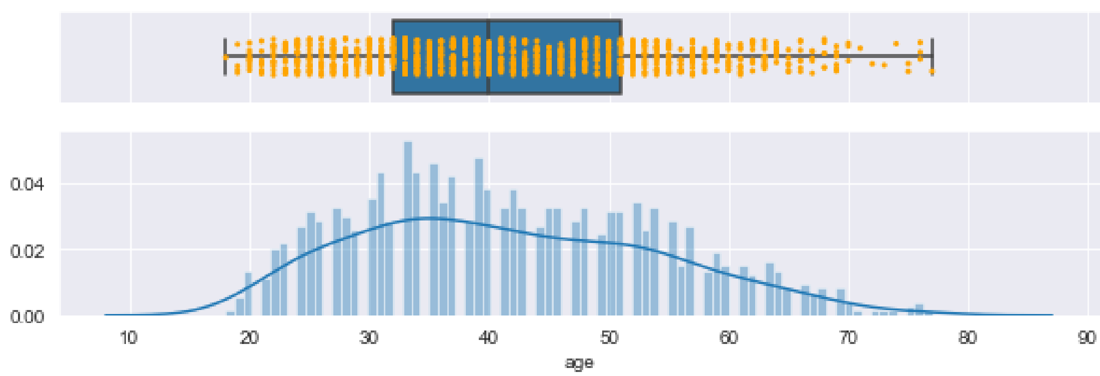


```
mean      35.526000
std       21.359812
min        1.000000
10%        7.000000
25%       17.000000
50%       34.000000
75%       54.000000
90%       66.000000
max       72.000000
skew       0.111692
var      456.241566
Name: tenure, dtype: float64
```

age (age in years)

In [15]:
```
column = 'age'
represent_distribution(df[column], varname=column)
print(describe_intervals[column])
```

```
Variable `age`
Number of observations: 1000
```



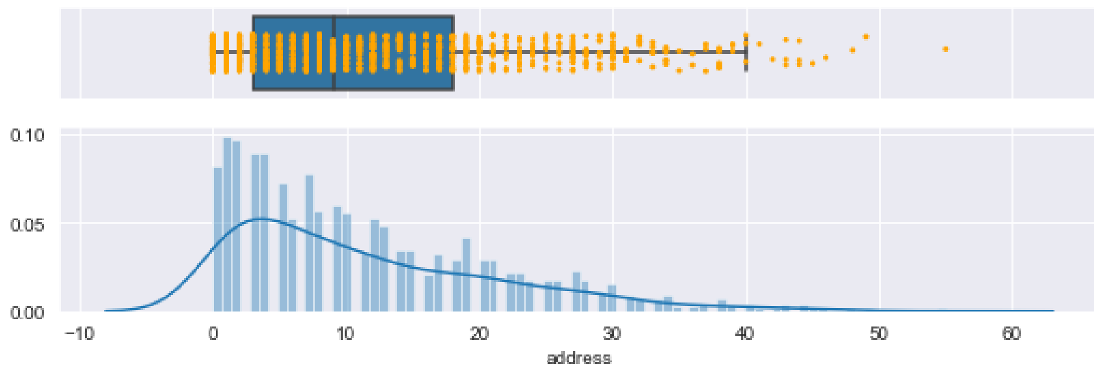

```
mean      41.684000
std       12.558816
min       18.000000
10%       26.000000
25%       32.000000
50%       40.000000
75%       51.000000
```

```
90%        59.000000
max        77.000000
skew        0.356128
var       157.723868
Name: age, dtype: float64
```

address (years at the current address)

```
In [16]: column = 'address'
         represent_distribution(df[column], varname=column)
         print(describe_intervals[column])
```

```
Variable `address`
Number of observations: 1000
```

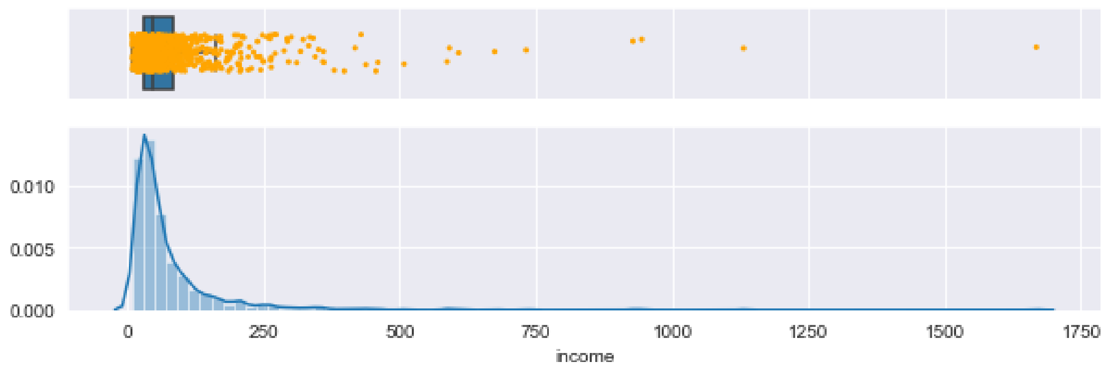



```
mean       11.551000
std        10.086681
min         0.000000
10%         1.000000
25%         3.000000
50%         9.000000
75%        18.000000
90%        26.100000
max        55.000000
skew        1.104586
var       101.741140
Name: address, dtype: float64
```

income (household income in Euro)

```
In [17]: column = 'income'
         represent_distribution(df[column], varname=column)
         print(describe_intervals[column])
```

```
Variable `income`
Number of observations: 1000
```




```
mean         77.535000
std         107.044165
min           9.000000
10%          21.000000
25%          29.000000
50%          47.000000
75%          83.000000
90%         155.400000
max        1668.000000
skew          6.633303
var       11458.453228
Name: income, dtype: float64
```

As with `longmon` we transform the variable `income` into `log(income)`:

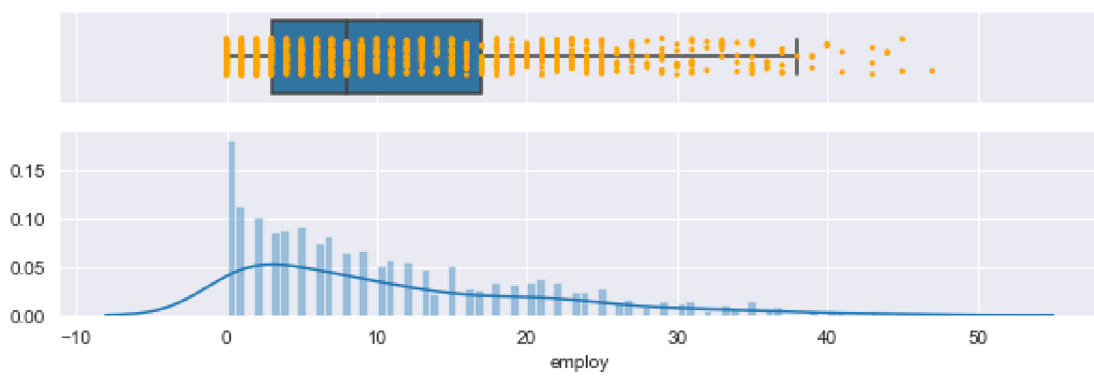
```
In [18]: df['log_{}'.format(column)] = np.log(df[column])
         column = 'log_{}'.format(column)
         represent_distribution(np.log(df[column]), varname=column)
```

```
Variable `log_income`
Number of observations: 1000
```

employ (years with the current employer)

```
In [19]: column = 'employ'
         represent_distribution(df[column], varname=column)
         print(describe_intervals[column])
```

```
Variable `employ`
Number of observations: 1000
```



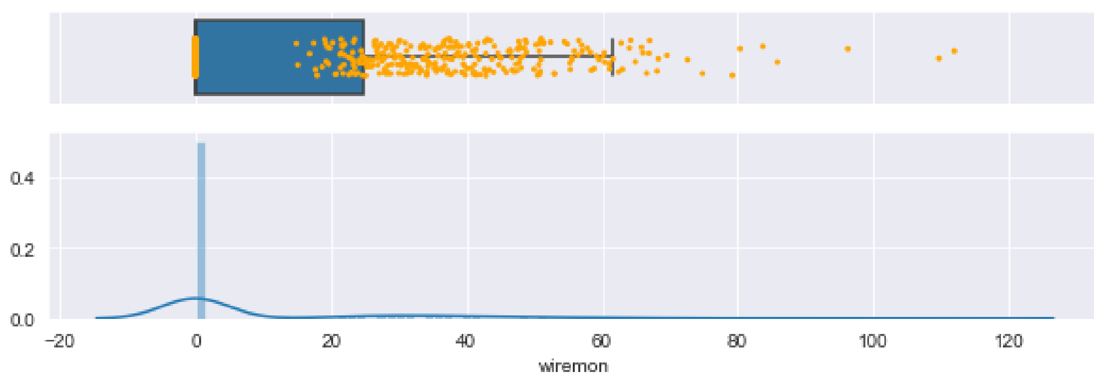

```
mean      10.987000
std       10.082087
min        0.000000
10%        0.000000
25%        3.000000
50%        8.000000
75%       17.000000
90%       25.000000
max       47.000000
```

```
skew        1.059457
var       101.648479
Name: employ, dtype: float64
```

wiremon (internet use last month)

```
In [20]: column = 'wiremon'
         represent_distribution(df[column], varname=column)
         print(describe_intervals[column])
```

```
Variable `wiremon`
Number of observations: 1000
```

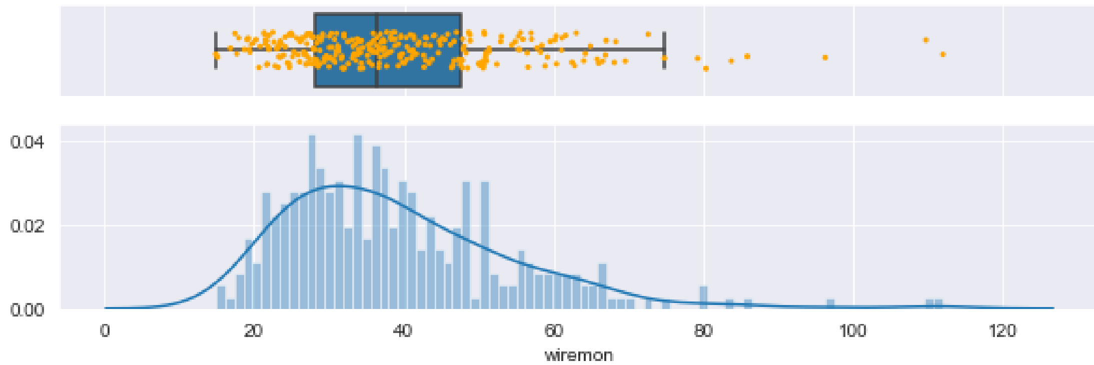



```
mean       11.583900
std        19.719426
min         0.000000
10%         0.000000
25%         0.000000
50%         0.000000
75%        24.712500
90%        42.110000
max       111.950000
skew        1.601274
var       388.855747
Name: wiremon, dtype: float64
```

Let's look at the distribution of usage for those who used the internet last month:

```
In [21]: represent_distribution(df[df[column]>0][column], varname=column+' > 0')
```

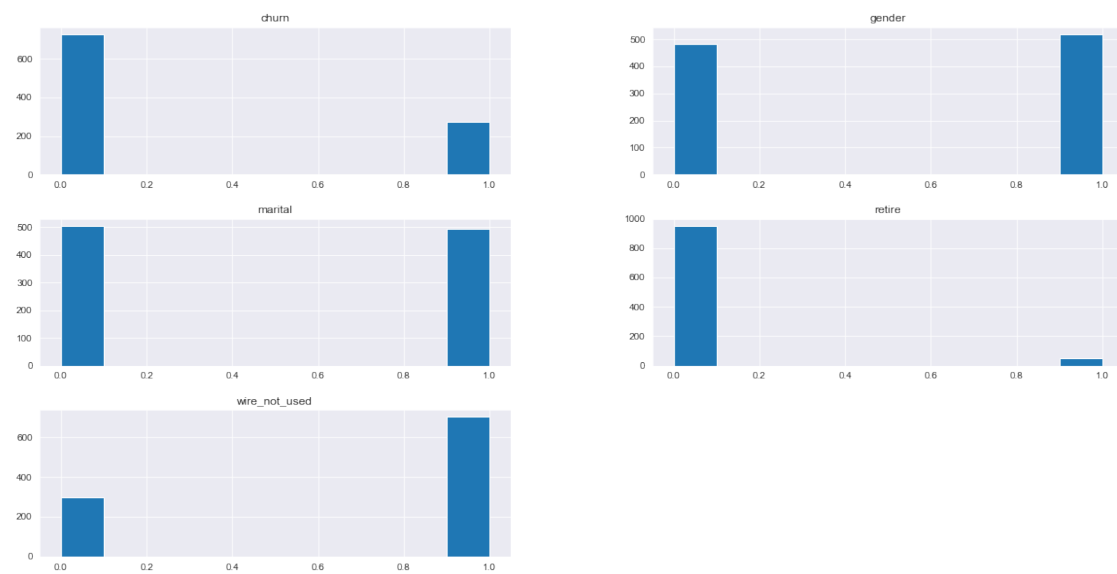
```
Variable `wiremon > 0`
Number of observations: 296
```



We can create a dummy variable representing, whether the client used or not the internet last month:

```
In [22]: df['wire_not_used'] = (df['wiremon']==0).astype(np.int)
         nominals.append('wire_not_used')
```

```
In [487]: df[nominals].hist(figsize=(20, 10))
          plt.show()
```




```
In [24]: df.head()
```

```
Out[24]:    tenure  age  marital  address  income  employ  retire  gender  longmon  \
         1      13   44        1        9      64       5       0       0     3.70
         2      11   33        1        7     136       5       0       0     4.40
```

```
3      68    52          1        24      116       29         0         1     18.15
4      33    33          0        12       33        0         0         1      9.45
5      23    30          1         9       30        2         0         0      6.30

   wiremon  churn  ed_college  ed_highschool  ed_postgr  ed_somecollege  \
1      0.0      1           1              0          0               0
2     35.7      1           0              0          1               0
3      0.0      0           0              0          0               0
4      0.0      1           0              1          0               0
5      0.0      0           0              0          0               0

   log_longmon  log_income  wire_not_used
1     1.308333    4.158883              1
2     1.481605    4.912655              0
3     2.898671    4.753590              1
4     2.246015    3.496508              1
5     1.840550    3.401197              1
```

1.3 3.

After making up your decision about the above two problems run a simple linear regression.

```
In [25]: def R_squared(y, y_hat):
             y = np.array(y).reshape(-1,1)
             y_hat = np.array(y_hat).reshape(-1,1)
             y_mean = y.mean()
             ESS = np.linalg.norm(y_hat - y_mean)**2
             TSS = np.linalg.norm(y - y_mean)**2
             return ESS/TSS

         def R_squared_adj(y, y_hat, K):
             y = np.array(y).reshape(-1,1)
             y_hat = np.array(y_hat).reshape(-1,1)
             N = len(y)
             y_mean = y.mean()
             RSS = np.linalg.norm(y - y_hat)**2
             TSS = np.linalg.norm(y - y_mean)**2
             return 1 - RSS * (N-1) / (TSS * (N-K-1))

         def AIC(y, y_hat, K):
             y = np.array(y).reshape(-1,1)
             y_hat = np.array(y_hat).reshape(-1,1)
             u = y - y_hat
             s2 = np.var(u, ddof=1)
             N = len(y)
             return np.log(s2) + 2 * K / N

         def BIC(y, y_hat, K):
```

```
    y = np.array(y).reshape(-1,1)
    y_hat = np.array(y_hat).reshape(-1,1)
    u = y - y_hat
    N = len(y)
    s2 = np.var(u, ddof=1)
    return np.log(s2) + K * np.log(N) / N
```

In [26]:
```
def get_X_y(df, regressand, regressors, out=True):
    if out:
        print('Regressand:', regressand)
        print('Regressors:', regressors)
    y = df[regressand]
    X = df[regressors]
    return X, y

def get_normalized_X_y(X, y, intervals_regressors):
    X_intervals_means = X[intervals_regressors].mean()
    X_intervals_stds = X[intervals_regressors].std(ddof=1)

    X_norm = X.copy()
    X_norm[intervals_regressors] = (X[intervals_regressors] - X_intervals_means) / X_i

    y_mean = y.mean()
    y_std = y.std(ddof=1)
    y_norm = (y - y_mean) / y_std

    return X_norm, y_norm, X_intervals_means, X_intervals_stds, y_mean, y_std

def get_LR_beta(X, y, out_scores=False):
    LR = linear_model.LinearRegression()
    LR.fit(X, y)
    beta_df = pd.DataFrame([LR.intercept_]+list(LR.coef_),
                           index=X.columns.insert(0, 1),
                           columns=['coef'])
    if out_scores:
        print('LR scores:')
        get_scores(LR, X, y)
    return LR, beta_df

def get_scores(LR, X, y):
    # y_hat = np.dot(sm.add_constant(X), beta_df['coef'].values.reshape(-1,1))
    y_hat = LR.predict(X)
    R2 = R_squared(y, y_hat)
    R2adj = R_squared_adj(y, y_hat, K)
    aic = AIC(y, y_hat, K)
    bic = BIC(y, y_hat, K)
    print('R^2 = {:.3f}\nR^2_adj = {:.3f}\nAIC = {:.3f}\nBIC = {:.3f}'.format(R2, R2ad
```

```
def get_standardized_coef(beta_df_orig, intervals_regressors, other_regressors):
    # For self-check
    beta_df = beta_df_orig.copy()
    beta_df['coef_st'] = beta_df['coef']
    beta_df.loc[intervals_regressors,'coef_st'] = beta_df.loc[intervals_regressors,'c
    beta_df.loc[other_regressors,'coef_st'] = beta_df.loc[other_regressors,'coef'] / y
    beta_df.loc[1,'coef_st'] = (beta_df.loc[1,'coef'] + np.sum(beta_df.loc[intervals_i
    return beta_df
```

```
In [707]: # Simple LR
          regressand = 'log_longmon'
          # with wiremon
          # regressors = sorted(list(set(df.columns).difference({'longmon', 'log_longmon', 'in
          # intervals_regressors = sorted(list(set(intervals).union({'log_income'}).difference

          # with wire_used
          regressors = sorted(list(set(df.columns).difference({'longmon', 'log_longmon', 'incom
          intervals_regressors = sorted(list(set(intervals).union({'log_income'}).difference({
          other_regressors = sorted(list(set(regressors).difference(intervals_regressors)))

          X, y = get_X_y(df, regressand, regressors)
          K = X.shape[1]
```

```
Regressand: log_longmon
Regressors: ['address', 'age', 'churn', 'ed_college', 'ed_highschool', 'ed_postgr', 'ed_somecol
```

Before runing LR we normalize X and y, and work only with it. As will be shown in Problem 5, it won't affect R^2 and we can get coefficients for the unnormalized data with corresponding linear transformtions.

```
In [709]: X, y, X_intervals_means, X_intervals_stds, y_mean, y_std = \
                                        get_normalized_X_y(X, y, intervals_regressors)
          X_ = sm.add_constant(X)

          LR, beta_df = get_LR_beta(X, y)
          beta_df['coef_abs'] = np.abs(beta_df['coef'])

          # Get standardized coefficients of LR
          # LR_st, beta_df = get_LR_beta(X_norm, y_norm, True)
          # beta_df['coef_st'] = beta_st['coef']
          # beta_df['coef_st_abs'] = np.abs(beta_df['coef_st'])
          print('\nStandardized LR coefficients:')
          print(beta_df.sort_values(by='coef_abs',  ascending=False)[['coef']])
          print()
          y_hat = LR.predict(X)
          print('R_squared: {:.3f}'.format(R_squared(y, y_hat)))
          print('R_squared_adj: {:.3f}'.format(R_squared_adj(y, y_hat, K)))
          print('AIC: {:.3f}'.format(AIC(y, y_hat, K)))
```

```
print('BIC: {:.3f}'.format(BIC(y, y_hat, K)))

LR = sm.OLS(y, X_)
LR_results = LR.fit()
# print(LR_results.summary())
```

```
Standardized LR coefficients:
                    coef
tenure           0.793197
retire           0.334490
ed_college       0.157259
1               -0.126201
ed_highschool    0.124676
ed_somecollege   0.108959
marital          0.098148
churn           -0.054113
address          0.051514
gender          -0.048634
age             -0.047336
employ           0.024508
log_income       0.023402
ed_postgr        0.012869
wire_not_used    0.008047

R_squared: 0.717
R_squared_adj: 0.713
AIC: -1.235
BIC: -1.167
```

If you wish to argue that `education` is insignicant and use the model with dummies then you have to check the simultaneous insignicance of all dummies which stem from the factor variable `ed`. Run a test for general linear hypothesis and conclude about the signicance of `ed`.

Variable `ed` seems to be important, let's check it:

```
In [31]: ed_vars = ['ed_somecollege', 'ed_college', 'ed_highschool', 'ed_postgr']

         def test_significance(LR_results, variabless):
             H0 = ' = '.join(variabless) + ' = 0'
             if len(variabless) > 1:
                 H0_text = 'all in {} simultaneously have no impact'.format(variabless)
                 H1_text = 'at least one in {} is significant'.format(variabless)
             else:
                 H0_text = '{} has no impact'.format(variabless)
                 H1_text = '{} is significant'.format(variabless)
             print('H0:', H0_text)
             print('H1:', H1_text)
```

```
        F_results = LR_results.f_test(H0)
        print('F = {:.4f}, p-value = {:.4f}'.format(F_results.fvalue[0,0], F_results.pvalu

    test_significance(LR_results, ed_vars)
```

```
H0: all in ['ed_somecollege', 'ed_college', 'ed_highschool', 'ed_postgr'] simultaneously have r
H1: at least one in ['ed_somecollege', 'ed_college', 'ed_highschool', 'ed_postgr'] is significa
F = 2.7484, p-value = 0.0272
```

The p-value is small enough so we decide to reject H0, which means that the variable `education` is significant.

1.4 4.

Provide an economic interpretation for the parameters of `address`, `ed`, and `retire`. Neglect the possible insignicance and keep in mind possible transformations of the variables.

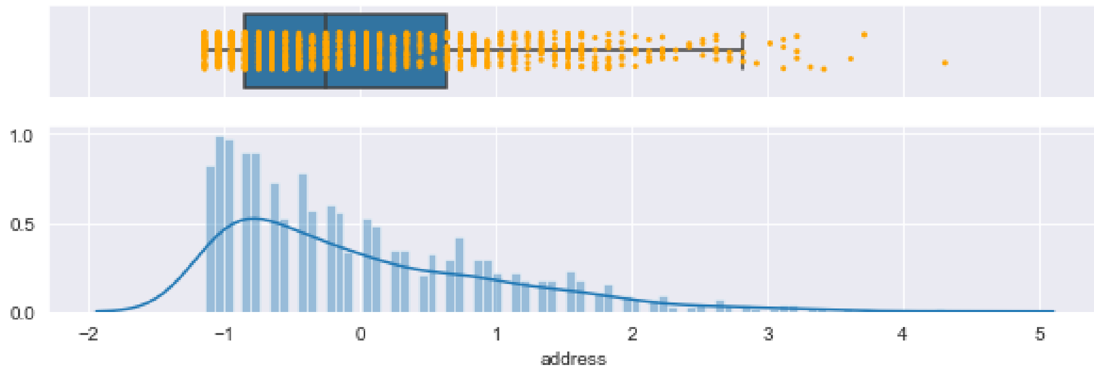
```
In [32]: beta_df.loc[['address', 'retire',
                      'ed_somecollege','ed_college',
                      'ed_highschool', 'ed_postgr'], :]\
         .sort_values(by='coef_abs', ascending=False)[['coef']]
```

Out[32]:

	coef
retire	0.334490
ed_college	0.157259
ed_highschool	0.124676
ed_somecollege	0.108959
address	0.051514
ed_postgr	0.012869

```
In [33]: print('Address mean: {:.2f}, std: {:.2f} (used for normalization)'.format(X_intervals_
         represent_distribution(X['address'], varname='normalized address')
         test_significance(LR_results, ['address'])
         print()
         test_significance(LR_results, ['retire'])
```

```
Address mean: -0.00, std: 1.00 (used for normalization)
Variable `normalized address`
Number of observations: 1000
```




```
H0: ['address'] has no impact
H1: ['address'] is significant
F = 4.6722, p-value = 0.0309

H0: ['retire'] has no impact
H1: ['retire'] is significant
F = 9.5569, p-value = 0.0020
```

The p-value for the first test with `address` is small so this variable is significant. This variable represents years at the current address but during the normalization 11.5 was subtracted from values and it was divided by 10 so now it's hard to interpret this variable. The p-value for the test with `retire` is also very small so this variable is significant. It is a nominal scaled variable representing whether a person is retired or not, and it wasn't changed. The coefficient 0.3345 tells that with the same other parameters change from 'not retired' to 'retired' would make us expect an increasing in normalized `log_longmon` by 0.3345. For `education` we already concluded that it is a significant variable. It represents type of education and, as was supposed, we see that indeed a change on different education levels causes a different change in target, and also the order of (sorted) coefficients is not the same as a natural order of education levels (where e.g. `college` should be between `highschool` and `postgr`) so the relation is more complex.

1.5 5.

Compute the 95% condence intervals for the parameters of `address` and `income` and provide its economic meaning. Relate the CIs to the tests of signicance, i.e. how would you use these intervals to decide about the signicance of the corresponding explanatory variables? The CIs are computed relying on the assumption, that the residuals follow normal distribution. Is this assumption fulfilled? Run an appropriate goodness-of-fit test.

```
In [34]: print('95%-CI:')
         CI = LR_results.conf_int(alpha=0.05).loc[['address', 'log_income'], :]\
                   .rename(columns={0:'lower', 1:'upper'})
```

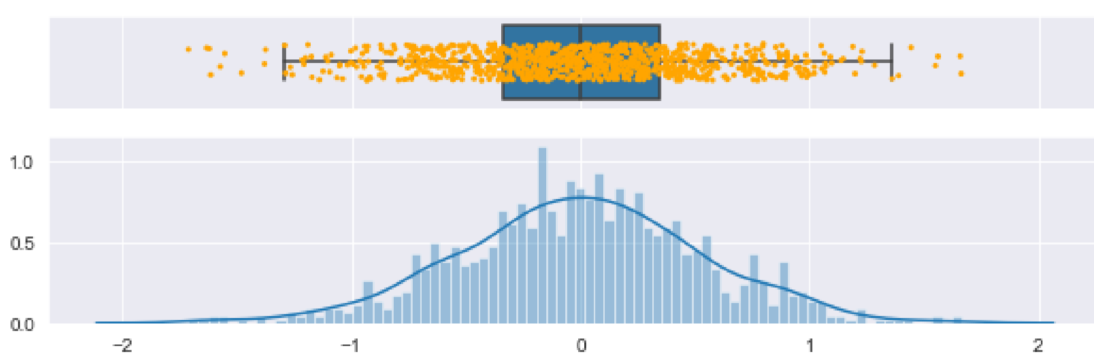
```
CI['length'] = CI.upper - CI.lower
print(CI)
print()
test_significance(LR_results, ['log_income'])
```

```
95%-CI:
               lower     upper    length
address     0.004746  0.098283  0.093537
log_income -0.033571  0.080375  0.113946

H0: ['log_income'] has no impact
H1: ['log_income'] is significant
F = 0.6497, p-value = 0.4204
```

The lower bound for the `log_income` is negative, the upper bound is positive. So zero value for the coefficient is inside 95% confidence interval which mean possible insignificance of the variable. The significance test results in quite large p-value 0.42, so we can reject the null-hypothesis (stating that the variable is significant) and drop `log_income`. We already concluded that `address` is significant. We see that the CI's length for this variable is less than for previous meaning it has less uncertainty.

```
In [35]: print('Residuals')
         res = LR_results.resid
         represent_distribution(res)
         print('Mean: {:.2e}'.format(res.mean()))
```

```
Residuals
Number of observations: 1000
```




```
Mean: -3.03e-15
```

The histogram of the residuals looks like it follows the normal distribution with zero-mean. To check the assumption on the normal distribution we will run Kolmogorov-Smirnov goodness-of-fit test.

```
In [48]: print('Kolmogorov-Smirnov test for residuals.')
         print('H0: res ~ Normal')
         print('H1: res !~ Normal')
         _, pvalue = stats.kstest(res/res.std(), 'norm')
         print('p-value: {:.2f}'.format(pvalue))
```

```
Kolmogorov-Smirnov test for residuals.
H0: res ~ Normal
H1: res !~ Normal
p-value: 0.85
```

With the p-value 0.85 we do not reject the hypothesis of normal distributed residuals. Therefore computing CI is reasonable.

1.6 6.

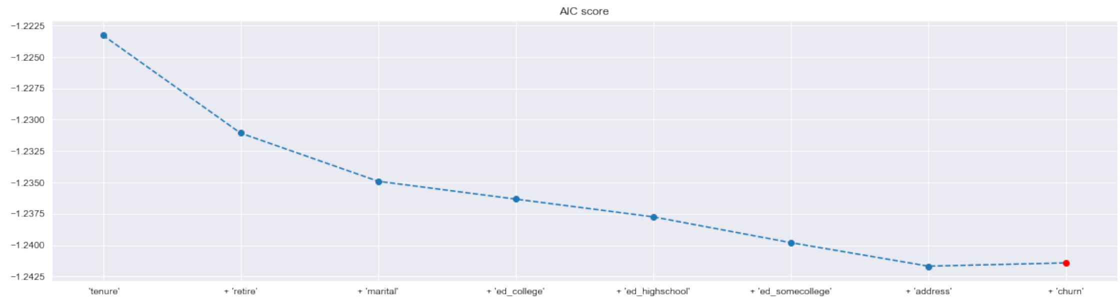
Many of the variable appear insignicant and we should nd the smallest model, which still has a good explanatory power. Choose this model using stepwise model selection (either based on the tests for R^2 or using AIC/BIC). Pick up the last step of the model selection procedure and explain in details how the method/approach works (or is implemented in your software). Work with this model in all the remaining steps.

```
In [147]: mpl.rcParams['figure.figsize'] = (20,5)
          print('In total {} regressors: {}'.format(len(regressors), regressors))

          def forward_selection(X, y, regressors, score_type, verbose=False, plot=True):
              score_fn = R_squared_adj if score_type == 'R^2_adj' else AIC if score_type == 'A
              regressors_selected, regressors_left = [], regressors.copy()
              scores_selected = []
              best_score = np.inf if score_type in ('AIC', 'BIC') else -np.inf
              score = best_score
              while regressors_left != [] and score==best_score:
                  regressor_score = []
                  for regressor in regressors_left:
                      X_selected = X.loc[:, regressors_selected+[regressor]]
                      LR = linear_model.LinearRegression()
                      LR.fit(X_selected, y)
                      y_hat = LR.predict(X_selected)
                      s = score_fn(y, y_hat, len(X_selected.columns))
                      regressor_score.append(s)
                      if verbose:
                          print(regressors_selected+[regressor],': {:.3f}'.format(s))
                  i = np.argmin(regressor_score) if score_type in ('AIC', 'BIC') \
                      else np.argmax(regressor_score)
                  score = regressor_score[i]
                  condition = (score < best_score) if score_type in ('AIC', 'BIC') \
                              else (score > best_score)
                  if condition:
```

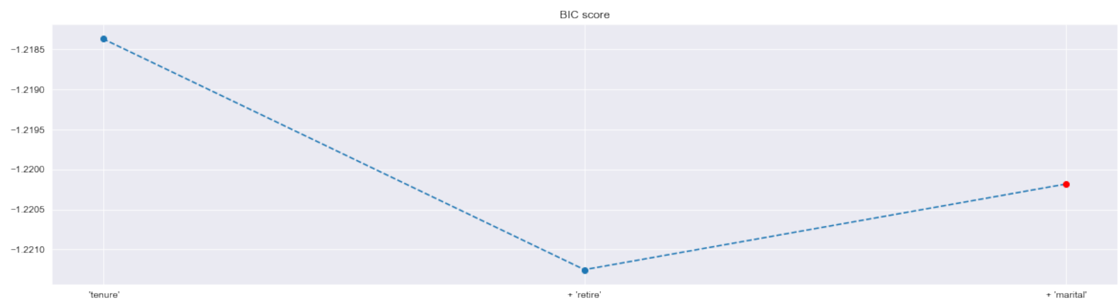
```
                if verbose:
                    print('> {} is chosen with score {}'.format(regressors_left[i], score
                best_score = score
                regressors_selected.append(regressors_left[i])
                scores_selected.append(score)
                del regressors_left[i]
        print('> In terms of {} ({}):\n{}'.format(score_type, len(regressors_selected), r
        if plot:
            xs = regressors_selected+[regressors_left[i]]
            xs = ['{} \'{}\''.format('' if i==0 else '+', x) for i, x in enumerate(xs)]
            plt.plot(xs, scores_selected+[score], '--')
            plt.scatter(xs[:-1], scores_selected)
            plt.scatter(xs[-1], score, color='r', zorder=10)
            plt.title('{} score'.format(score_type))
            plt.show()
        return regressors_selected

    print('\nThe best model (forward selection):')
    forward_selection(X, y, regressors, 'R^2_adj')
    regressors_selected = forward_selection(X, y, regressors, 'AIC')
    forward_selection(X, y, regressors, 'BIC');
```

```
In total 14 regressors: ['address', 'age', 'churn', 'ed_college', 'ed_highschool', 'ed_postgr'

The best model (forward selection):
> In terms of R^2_adj (11):
['tenure', 'retire', 'marital', 'ed_college', 'ed_highschool', 'ed_somecollege', 'address', 'c
```


```
> In terms of AIC (7):
['tenure', 'retire', 'marital', 'ed_college', 'ed_highschool', 'ed_somecollege', 'address']
```




```
> In terms of BIC (2):
['tenure', 'retire']
```



The result of forward model selection using AIC was chosen as a middle ground. The selected regressors are: `tenure`, `retire`, `marital`, `ed_college`, `ed_highschool`, `ed_somecollege`, `address`.

In [148]:
```
intervals_regressors = list(set(regressors_selected).intersection(set(intervals_regre
other_regressors = list(set(regressors_selected).intersection(set(other_regressors))
X = X.loc[:, regressors_selected]
X_ = sm.add_constant(X)
LR = sm.OLS(y, X_)
LR_results = LR.fit()
print(LR_results.summary())
```

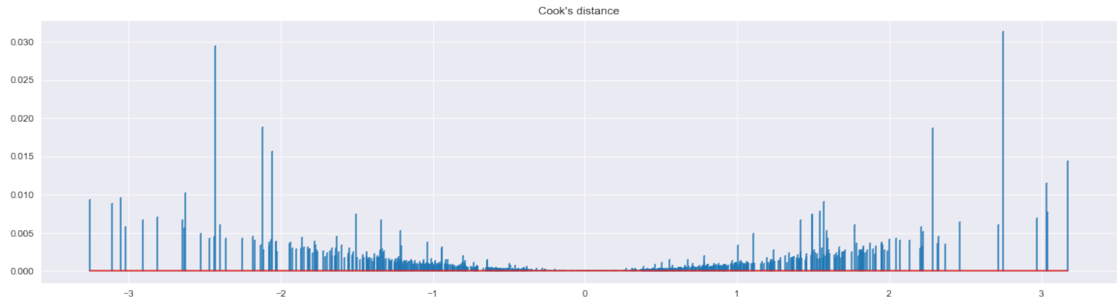
```
                            OLS Regression Results
==============================================================================
Dep. Variable:             log_longmon   R-squared:                       0.715
Model:                             OLS   Adj. R-squared:                  0.713
Method:                  Least Squares   F-statistic:                     355.7
Date:                 Tue, 26 Feb 2019   Prob (F-statistic):         2.36e-265
Time:                         13:38:25   Log-Likelihood:                -790.60
No. Observations:                 1000   AIC:                             1597.
```

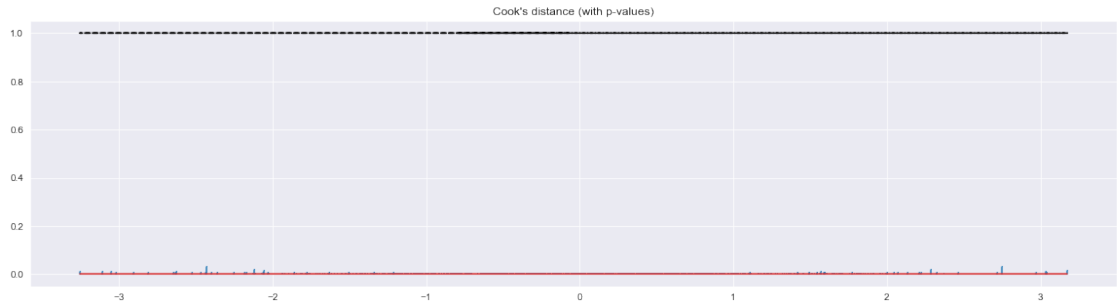
```
Df Residuals:                     992   BIC:                             1636.
Df Model:                           7
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const             -0.1508      0.038     -3.969      0.000      -0.225      -0.076
tenure             0.8079      0.020     39.798      0.000       0.768       0.848
retire             0.2546      0.084      3.019      0.003       0.089       0.420
marital            0.0927      0.035      2.675      0.008       0.025       0.161
ed_college         0.1515      0.048      3.138      0.002       0.057       0.246
ed_highschool      0.1229      0.046      2.698      0.007       0.034       0.212
ed_somecollege     0.1061      0.050      2.136      0.033       0.009       0.204
address            0.0405      0.021      1.962      0.050   -4.99e-06       0.081
==============================================================================
Omnibus:                        2.592   Durbin-Watson:                   1.944
Prob(Omnibus):                  0.274   Jarque-Bera (JB):                2.634
Skew:                          -0.054   Prob(JB):                        0.268
Kurtosis:                       3.227   Cond. No.                         6.24
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

1.7 7.

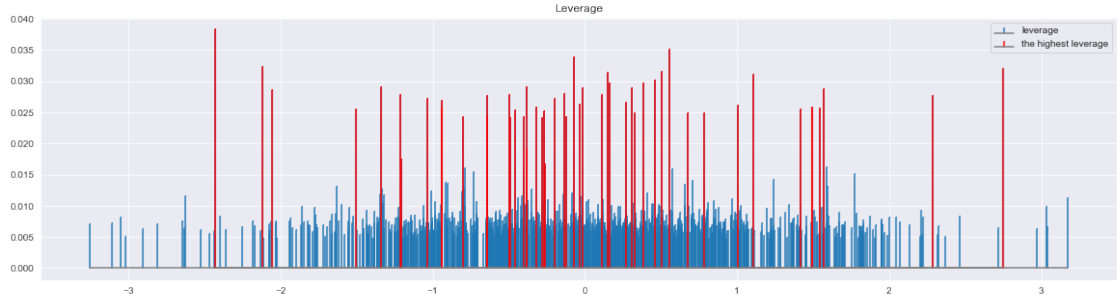
Sometimes data contains outliers which induces bias in the parameter estimates. Check for outliers using Cook's distance and leverage. Have a closer look at the observation with the highest leverage (regardless if it is classied as an outlier or not). What makes this observation so outstanding (you may have a look at Box-plots for interval scaled variables or at the frequencies for binary/ordinal variables?

In [207]:
```python
from statsmodels.stats.outliers_influence import OLSInfluence

influence = OLSInfluence(LR_results).summary_frame()
pvals = LR_results.get_influence().cooks_distance[1]
plt.stem(influence.standard_resid, influence.cooks_d, markerfmt=",")
plt.title('Cook\'s distance')
plt.show()
plt.stem(influence.standard_resid, influence.cooks_d, markerfmt=",")
plt.plot(influence.standard_resid, pvals, 'k--')
plt.title('Cook\'s distance (with p-values)')
plt.show()
```

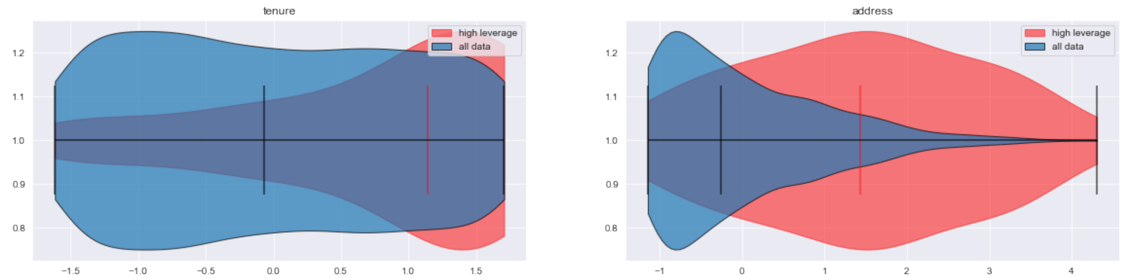


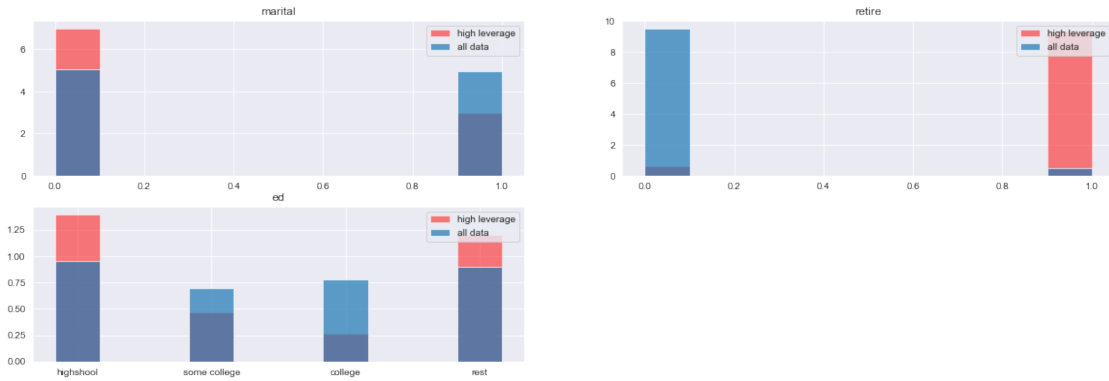


In [216]:
```
plt.title('Leverage')
leverage = influence.hat_diag
plt.stem(influence.standard_resid, leverage, markerfmt=",", basefmt='gray')

n = 50
ind = leverage.index[(-leverage).argsort()][:n]
plt.stem(influence.standard_resid.loc[ind], leverage.loc[ind],
         markerfmt="r,", linefmt='r', basefmt='gray')

plt.legend(['leverage', 'the highest leverage'])
plt.show()
```



```
In [458]: temp = X.copy()
          temp['HighLeverage'] = False
          temp.loc[ind, 'HighLeverage'] = True
          temp['ed'] = (1*temp.ed_highschool + 2*temp.ed_somecollege + 3*temp.ed_college + 3) 

          mpl.rcParams['figure.figsize'] = (20,10)
          for i, r in enumerate(intervals_regressors):
              plt.subplot(len(intervals_regressors)//2+1, 2, i+1)
              plt.title(r)
              p1 = plt.violinplot(temp[temp['HighLeverage']==True][r],
                                       showextrema=False, showmedians=True, vert=False)
              p2 = plt.violinplot(temp[r],
                                  showextrema=True, showmedians=True, vert=False)
              for pc in p1['bodies']+[p1['cmedians']]:
                  pc.set_facecolor('red')
                  pc.set_edgecolor('red')
                  pc.set_alpha(0.5)
              for pc in p2['bodies']+[p2['cmedians']] + [p2['cbars']]+ [p2['cmins']]+ [p2['cma
                  pc.set_facecolor('C0')
                  pc.set_edgecolor('k')
                  pc.set_alpha(0.7)
              p1['bodies'][0].set_label('high leverage')
              p2['bodies'][0].set_label('all data')
              plt.legend()
          plt.show()

          for i, r in enumerate([r for r in other_regressors + ['ed'] if r[:3]!='ed_']):
              plt.subplot(len(other_regressors)//2+1, 2, i+1)
              plt.title(r)
              plt.hist(temp[temp['HighLeverage']==True][r], alpha=0.5, density=True, color='r'
              plt.hist(temp[r], alpha=0.7, density=True)
              plt.legend(['high leverage', 'all data'], loc=1)
              if r=='ed':
                  plt.xticks([0.15,1.05,1.95,2.85], ['highshool', 'some college', 'college', '
          plt.show()
```





One can se from the distributions that the observations with the highest leverage have characteristics such as high tenure and long period at the current address. It is more often when the 'outlier' person is single, gets lower/no education and almost all of them are retired,

1.8 8.

Frequently data is missing. Pick up 5 rows in the data set and delete the value for address. Implement at least two approaches to ll in these values. Write down the corresponding formulas/model and give motivation for your approach. If you use standard routines then check how exactly the data imputation is implemented. How would you proceed if the value of the binary variable `retire` is missing? Implementation is not required.

```
In [480]: # Missing completely at random
          X_miss = X.copy()
          np.random.seed(100)
          m = 5
          ind_to_delete = np.random.randint(1,len(X_miss)+1, size=m)
          X_miss.loc[ind_to_delete,'address'] = np.nan
          X_miss = get_normalized_X_y(X_miss, y, intervals_regressors)[0]
          X_miss.loc[ind_to_delete,:]
```

```
Out[480]:        tenure  retire  marital  ed_college  ed_highschool  ed_somecollege  \
          521  0.490360       0        1           0              0               0
          793  0.256276       0        1           0              1               0
          836 -1.195048       0        1           0              1               0
          872  1.613966       0        0           0              0               0
          856  0.677628       0        0           0              0               0

               address
          521      NaN
          793      NaN
          836      NaN
          872      NaN
          856      NaN
```

1. Drop the rows with missing data:

```
In [481]: X1 = X_miss.drop(ind_to_delete, 0)
          y1 = y.drop(ind_to_delete, 0)

          LR1, beta_df1 = get_LR_beta(X1, y1, True)
```

```
LR scores:
R^2 = 0.715
R^2_adj = 0.711
AIC = -1.229
BIC = -1.160
```

2. Hot imputation: for each observation with missing address get the `k=10` closest (by variables except `address`) observations without missing address and randomly choose one of them to replace missing value:

```
In [483]: np.random.seed(100)
          X2 = X_miss.copy()
          k = 10
          for i in range(m):
              other_values = list(X2.loc[ind_to_delete[i], :][other_regressors])
              X2[np.min(X2[other_regressors] - other_values == 0, 1)]

              intervals_regressors_ = sorted(list(set(intervals_regressors).difference({'addres
              intervals_values = list(X2.loc[ind_to_delete[i], :][intervals_regressors_])
              aux_df = (np.square(X2[intervals_regressors_] - intervals_values)).drop(ind_to_de
              aux_df['mean'] = np.mean(aux_df[intervals_regressors_], 1)
              ind_to_replace = np.random.choice(aux_df.sort_values(by='mean').head(k).index)
          #     print(ind_to_delete[i], '<-', ind_to_replace)
              X2.loc[ind_to_delete[i], 'address'] = X2.loc[ind_to_replace,'address']

          LR2, beta_df2 = get_LR_beta(X2, y, True)
```

```
LR scores:
R^2 = 0.715
R^2_adj = 0.711
AIC = -1.227
BIC = -1.158
```

3. Mean imputation: replace missing values with the mean of the variable:

```
In [484]: X3 = X_miss.copy()
          X3.loc[ind_to_delete, 'address'] = np.mean(X3['address'])
          LR3, beta_df3 = get_LR_beta(X3, y, True)
```

```
LR scores:
R^2 = 0.715
```

```
R^2_adj = 0.711
AIC = -1.227
BIC = -1.159
```

4. Regression imputation: replace missing values with the forecast using the Linear Regression model with `address` as a dependent variable, and all the others (except actual regressand `log_longmon`) as regressors:

```
In [485]: X4 = X_miss.copy()

          XX = X_miss.drop(ind_to_delete, 0).drop('address',1)
          Xy = X_miss.drop(ind_to_delete, 0)['address']

          XLR, _ = get_LR_beta(XX, Xy)
          X4.loc[ind_to_delete, 'address'] = XLR.predict(X_miss.loc[ind_to_delete,:].drop('addr
          LR4, beta_df4 = get_LR_beta(X4, y, True)
```

```
LR scores:
R^2 = 0.715
R^2_adj = 0.711
AIC = -1.227
BIC = -1.159
```

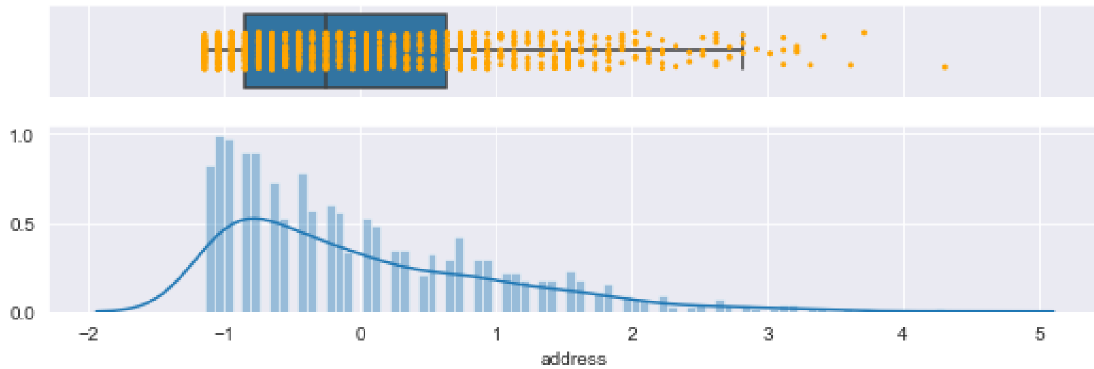
All the methods seem to work well. Droping 5 rows with missing values may not affect the result too much but in case of more frequent missing values we may loose useful information since it may be contained in the droped rows. If the value of the binary variable `retire` is missing I would suggest to use hot imputation or median imputation or forecast using the logistic regression model.

1.9 9.

Now we look at the model assumptions. The variable `address` seems to be very significant. However, if we look at the residuals we observe that the variance of the residuals is rather dierent for dierent values of address. Run the Bartlett's test and compute the FGLS estimators assuming an exponential relationship between the variance of residuals and address. Compare the results with the original model. Explain the advantages of the (F)GLS estimation.

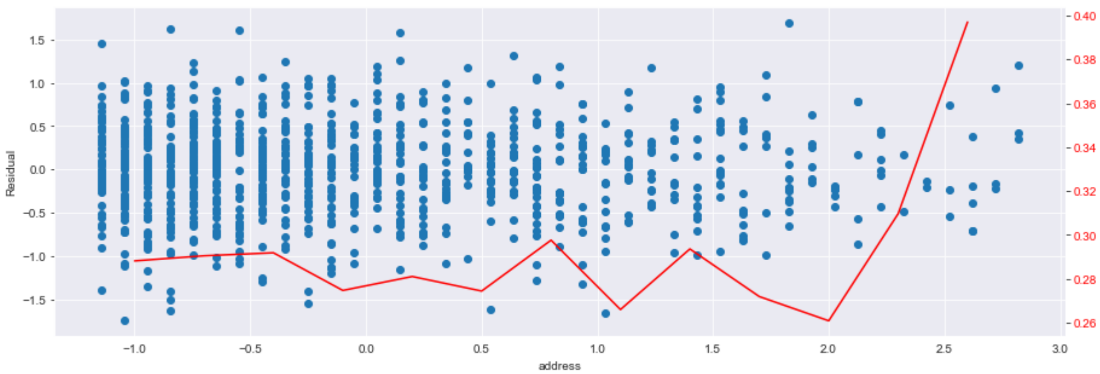
```
In [633]: represent_distribution(X['address'], varname='address')
          q25, q75 = p_quantile(X.address, 0.25), p_quantile(X.address, 0.75)
          upper_bound = q75 + 1.5*(q75 - q25)
          print('We will drop observations with address value > {:.2f} (due to a small number o
```

```
Variable `address`
Number of observations: 1000
```



We will drop observations with address value > 2.87 (due to a small number of observations)

```
In [639]: mpl.rcParams['figure.figsize'] = (15,5)

          temp = X[['address']]
          temp['resid'] = LR_results.resid
          temp = temp[temp.address<upper_bound]

          vs = []
          delta = 0.3
          l = np.arange(-1,3,delta)
          for i,j in zip(l, np.arange(0,4,delta)):
              v = temp[temp.address > i][temp.address < j].resid.var()
              vs.append(v)

          fig, ax1 = plt.subplots()
          ax1.scatter(temp.address, temp.resid, zorder=10)
          ax1.set_ylabel('Residual')
          ax1.set_xlabel('address')

          ax2 = ax1.twinx()
          ax2.grid('off')
          ax2.plot(l,vs, 'r', label=' of residuals')
          ax2.set_ylabel('Variance', color='r')
          ax2.tick_params(axis='y', labelcolor='r')
          plt.show()
```




```
In [657]: print('Bartlett\'s Test:')
          print('H0: sigma_1^2 = ... = sigma_2^2')
          print('H1: sigma_i^2 != sigma_j^2 for at least one pair (i,j)')
          stats.bartlett(*[r[1].values for r in temp.groupby('address').resid])
```

```
Bartlett's Test:
H0: sigma_1^2 = ... = sigma_2^2
H1: sigma_i^2 != sigma_j^2 for at least one pair (i,j)
```

```
Out[657]: BartlettResult(statistic=38.69457145789063, pvalue=0.5289947719634349)
```

With the p-value is 0.529 we can not reject the null hypothesis. So we don't have enough arguments to reject that the variances are the same.

```
In [692]: multiplier = sm.OLS(np.log(temp.resid**2), sm.add_constant(temp.address)).fit().param
          print('The LR result: sigma_i^2 = sigma^2 * exp({:.3f} * address_i)\n'.format(multipl

          w = np.exp(multiplier * X.address)

          LR_OLS = sm.OLS(y, X_)
          LR_OLS_results = LR_OLS.fit()
          print(LR_OLS_results.summary())
          print()

          LR_FGLS = sm.GLS(y, X_, weights=w)
          LR_FGLS_results = LR_FGLS.fit()
          print(LR_FGLS_results.summary())
```

```
The LR result: sigma_i^2 = sigma^2 * exp(0.098 * address_i)

                            OLS Regression Results
==============================================================================
Dep. Variable:             log_longmon   R-squared:                       0.715
```

```
Model:                            OLS   Adj. R-squared:                  0.713
Method:                 Least Squares   F-statistic:                     355.7
Date:                Tue, 26 Feb 2019   Prob (F-statistic):          2.36e-265
Time:                        17:25:40   Log-Likelihood:                -790.60
No. Observations:                1000   AIC:                             1597.
Df Residuals:                     992   BIC:                             1636.
Df Model:                           7
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const             -0.1508      0.038     -3.969      0.000      -0.225      -0.076
tenure             0.8079      0.020     39.798      0.000       0.768       0.848
retire             0.2546      0.084      3.019      0.003       0.089       0.420
marital            0.0927      0.035      2.675      0.008       0.025       0.161
ed_college         0.1515      0.048      3.138      0.002       0.057       0.246
ed_highschool      0.1229      0.046      2.698      0.007       0.034       0.212
ed_somecollege     0.1061      0.050      2.136      0.033       0.009       0.204
address            0.0405      0.021      1.962      0.050   -4.99e-06       0.081
==============================================================================
Omnibus:                        2.592   Durbin-Watson:                   1.944
Prob(Omnibus):                  0.274   Jarque-Bera (JB):                2.634
Skew:                          -0.054   Prob(JB):                        0.268
Kurtosis:                       3.227   Cond. No.                         6.24
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

                            GLS Regression Results
==============================================================================
Dep. Variable:            log_longmon   R-squared:                       0.723
Model:                            GLS   Adj. R-squared:                  0.721
Method:                 Least Squares   F-statistic:                     370.2
Date:                Tue, 26 Feb 2019   Prob (F-statistic):          1.66e-271
Time:                        17:25:40   Log-Likelihood:                -790.60
No. Observations:                1000   AIC:                             1597.
Df Residuals:                     992   BIC:                             1636.
Df Model:                           7
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const             -0.1508      0.038     -3.969      0.000      -0.225      -0.076
tenure             0.8079      0.020     39.798      0.000       0.768       0.848
retire             0.2546      0.084      3.019      0.003       0.089       0.420
marital            0.0927      0.035      2.675      0.008       0.025       0.161
ed_college         0.1515      0.048      3.138      0.002       0.057       0.246
```

```
ed_highschool       0.1229      0.046      2.698      0.007       0.034       0.212
ed_somecollege      0.1061      0.050      2.136      0.033       0.009       0.204
address             0.0405      0.021      1.962      0.050   -4.99e-06       0.081
==============================================================================
Omnibus:                        2.592   Durbin-Watson:                   1.944
Prob(Omnibus):                  0.274   Jarque-Bera (JB):                2.634
Skew:                          -0.054   Prob(JB):                        0.268
Kurtosis:                       3.227   Cond. No.                         6.24
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The advantage of the feasible GLS is that the parameter estimator is more efficient than that of OLS. This advantage is seen only when the weights of the error covariance matrix are correlated with data. If the weights are known, we have the best linear unbiased estimator. In our case and more often it is unknown. Still with the estimated weights (FGLS) we may get more efficient parameters.

Here we got identical results which is expectable since with the Bartlett's test we couldn't reject the hypothesis that the variances are the same.

1.10 10.

Compute the White estimator of covariance matrix of the OLS estimators. Run the t-tests and compare the results with the original model. Explain the advantages of the White estimator for the variance.

```
In [705]: print(LR_OLS.fit().summary())
          print()
          print(LR_OLS.fit(cov_type='HC0').summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             log_longmon   R-squared:                       0.715
Model:                             OLS   Adj. R-squared:                  0.713
Method:                  Least Squares   F-statistic:                     355.7
Date:                 Tue, 26 Feb 2019   Prob (F-statistic):         2.36e-265
Time:                         17:32:12   Log-Likelihood:                -790.60
No. Observations:                 1000   AIC:                             1597.
Df Residuals:                      992   BIC:                             1636.
Df Model:                            7
Covariance Type:             nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const             -0.1508      0.038     -3.969      0.000      -0.225      -0.076
tenure             0.8079      0.020     39.798      0.000       0.768       0.848
```

```
retire              0.2546      0.084      3.019      0.003       0.089       0.420
marital             0.0927      0.035      2.675      0.008       0.025       0.161
ed_college          0.1515      0.048      3.138      0.002       0.057       0.246
ed_highschool       0.1229      0.046      2.698      0.007       0.034       0.212
ed_somecollege      0.1061      0.050      2.136      0.033       0.009       0.204
address             0.0405      0.021      1.962      0.050   -4.99e-06       0.081
==============================================================================
Omnibus:                        2.592   Durbin-Watson:                   1.944
Prob(Omnibus):                  0.274   Jarque-Bera (JB):                2.634
Skew:                          -0.054   Prob(JB):                        0.268
Kurtosis:                       3.227   Cond. No.                         6.24
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

                            OLS Regression Results
==============================================================================
Dep. Variable:            log_longmon   R-squared:                       0.715
Model:                            OLS   Adj. R-squared:                  0.713
Method:                 Least Squares   F-statistic:                     324.6
Date:                Tue, 26 Feb 2019   Prob (F-statistic):          1.88e-251
Time:                        17:32:12   Log-Likelihood:                -790.60
No. Observations:                1000   AIC:                             1597.
Df Residuals:                     992   BIC:                             1636.
Df Model:                           7
Covariance Type:                  HC0
==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const             -0.1508      0.041     -3.649      0.000      -0.232      -0.070
tenure             0.8079      0.021     37.769      0.000       0.766       0.850
retire             0.2546      0.087      2.916      0.004       0.084       0.426
marital            0.0927      0.035      2.644      0.008       0.024       0.161
ed_college         0.1515      0.048      3.129      0.002       0.057       0.246
ed_highschool      0.1229      0.047      2.625      0.009       0.031       0.215
ed_somecollege     0.1061      0.051      2.073      0.038       0.006       0.206
address            0.0405      0.021      1.934      0.053      -0.001       0.081
==============================================================================
Omnibus:                        2.592   Durbin-Watson:                   1.944
Prob(Omnibus):                  0.274   Jarque-Bera (JB):                2.634
Skew:                          -0.054   Prob(JB):                        0.268
Kurtosis:                       3.227   Cond. No.                         6.24
==============================================================================

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
```

The white estimator corrected the confidence intervals since it is computed based on the variance of the parameters — white estimator of the covariance matrix based on the residuals and is used in computing the variance of the parameters. Also in the significance tests the statistic values changed and the p-values increased, but not significantly.

Problem 5: further issues

(a) Let $y_i = \beta_0 + \sum_{k=1}^{K} \beta_k x_{ik} + u_i,\ i = \overline{1, N}$ – a linear regression model for y_i.

$$\hat{y}_i = \hat{\beta}_0 + \sum_{k=1}^{K} \beta_k x_{ik} \qquad \hat{u}_i = y_i - \hat{y}_i$$

Let $y_i^* = y_i + 10$. To get the same least squares of residuals:

$$\hat{u}_i^* = \hat{u}_i = (y_i + 10 - 10) - \hat{y}_i = (y_i + 10) - (\hat{y}_i + 10) = y_i^* - \hat{y}_i^*$$

we should have $\hat{y}_i^* = \hat{y}_i + 10 = (\hat{\beta}_0 + 10) + \sum_{k=1}^{K} \beta_k x_{ik}$. So $\hat{\beta}_0^* = \hat{\beta}_0 + 10$

$$\bar{y}^* = \frac{\sum_{i=1}^{N} y_i^*}{N} = \bar{y} + 10 \qquad s_{y^*}^2 = \frac{\sum_{i=1}^{N} (y_i^* - \bar{y}^*)^2}{N-1} = \frac{\sum_{i=1}^{N} (y_i - \bar{y})^2}{N-1} = s_y^2$$

$$\Rightarrow\ R^{*2} = 1 - \frac{\sum_{i=1}^{N} \hat{u}_i^{*2}}{s_{y^*}^2} = 1 - \frac{\sum_{i=1}^{N} \hat{u}_i^2}{s_y^2} = R^2$$

Let's also show it in matrix form:

$$\mathbf{y} = X\boldsymbol{\beta} + \mathbf{u}, \quad \hat{\mathbf{y}} = X\hat{\boldsymbol{\beta}}, \quad \hat{\mathbf{u}} = \mathbf{y} - \hat{\mathbf{y}}$$

$$X \in N \times (K+1), \quad \mathbf{y}, \hat{\mathbf{y}}, \mathbf{u} \in \mathbb{R}^N, \quad \boldsymbol{\beta}, \hat{\boldsymbol{\beta}} \in \mathbb{R}^{K+1}$$

The paramaters from OLS:

$$\hat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{y}$$

Let $(X^\top X)^{-1} = \mathbf{A} = \begin{pmatrix} \mathbf{a}_0^\top \\ \vdots \\ \mathbf{a}_K^\top \end{pmatrix}$. Then $\hat{\boldsymbol{\beta}} = \mathbf{A} X^\top \mathbf{y},\ \hat{\beta}_0 = \mathbf{a}_0^\top X^\top \mathbf{y}$.

Let $y_i^* = y_i + 10$.

$$\mathbf{y}^* = \mathbf{y} + 10 \cdot \mathbf{1}, \quad \mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^N$$

$$\hat{\boldsymbol{\beta}}^* = \mathbf{A} X^\top \mathbf{y}^* = \mathbf{A} X^\top \mathbf{y} + 10 \mathbf{A} X^\top \mathbf{1} \qquad \hat{\beta}_0^* = \hat{\beta}_0 + 10\ \mathbf{a}_0^\top X^\top \mathbf{1}$$

$$\hat{\mathbf{y}}^* = X\hat{\boldsymbol{\beta}}^* = X\hat{\boldsymbol{\beta}} + 10 X \mathbf{A} X^\top \mathbf{1} = \hat{\mathbf{y}} + 10 X \mathbf{A} X^\top \mathbf{1}$$

Let's prove that $X\mathbf{A}X^\top \mathbf{1} = \mathbf{1}$. Indeed, since $P = X(X^\top X)^{-1} X^\top$ is the projector onto the column space $\mathcal{C}(X)$ and the first column is $\mathbf{1}$: $P\mathbf{1} = \mathbf{1}$. So:

$$\hat{\mathbf{y}}^* = \hat{\mathbf{y}} + 10 \cdot \mathbf{1}$$

From this one can also see that $\mathbf{A}X^\top \mathbf{1} = \mathbf{e}_1 = \begin{pmatrix}1 & 0 & \cdots & 0\end{pmatrix}^\top \in \mathbb{R}^{K+1}$ as the coefficients for the linear combination of columns of X to get $\mathbf{1}$, and hence:

$$\hat{\boldsymbol{\beta}}^* = \mathbf{A}X^\top \mathbf{y} + 10\mathbf{e}_1 \quad \hat{\beta}_0^* = \hat{\beta}_0 + 10$$

$$\hat{\mathbf{u}}^* = \mathbf{y}^* - \hat{\mathbf{y}}^* = \mathbf{y} + 10 \cdot \mathbf{1} - \hat{\mathbf{y}} - 10 \cdot \mathbf{1} = \hat{\mathbf{u}}$$

$$R^{*2} = 1 - \frac{\|\hat{\mathbf{u}}^*\|^2}{s_{y^*}^2} = 1 - \frac{\|\hat{\mathbf{u}}\|^2}{s_y^2} = R^2$$

Answer: $\hat{\beta}_0^* - \hat{\beta}_0 = 10$;
$R^{*2} = R^2$;

Now let $y_i^* = y_i - 10$. Similarly $\hat{\boldsymbol{\beta}}^* = \hat{\boldsymbol{\beta}} - 10\mathbf{e}_1$,

$$\hat{\beta}_0^* - \hat{\beta}_0 = -10 \qquad R^{*2} = R^2$$

So for δ_y we have $y_i^* = y_i + \delta_y$: $\hat{\beta}_0^* = \hat{\beta}_0 + \delta_y, \quad \hat{\beta}_i^* = \hat{\beta}_i \ (i = \overline{1, K}), \quad R^{*2} = R^2$

(b) Now let's continue with a linear regression model:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_K x_{iK}$$

If we change say all x_{i1} by δ_1: $x_{i1}^* = x_{i1} + \delta_1$, similarly from the OLS we should get the same $\hat{y}_i^* = \hat{y}_i$, since the change of x_{i1} is linear. So:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_K x_{iK} = \hat{\beta}_0 + \hat{\beta}_1 (x_{i1} + \delta_1 - \delta_1) + \cdots + \hat{\beta}_K x_{iK} = \hat{\beta}_0 + \hat{\beta}_1 x_{i1}^* - \hat{\beta}_1 \delta_1 + \cdots + \hat{\beta}_K x_{iK} =$$

$$= (\hat{\beta}_0 - \hat{\beta}_1 \delta_1) + \hat{\beta}_1 x_{i1}^* + \cdots + \hat{\beta}_K x_{iK} = \hat{\beta}_0^* + \hat{\beta}_1^* x_{i1}^* + \cdots + \hat{\beta}_K^* x_{iK}$$

$$\hat{\beta}_0^* = \hat{\beta}_0 - \hat{\beta}_1 \delta_1$$

So in general for $x_{ik}^* = x_{ik} + \delta_k$:

$$\hat{\beta}_0^* = \hat{\beta}_0 - \sum_{k=1}^{K} \hat{\beta}_k \delta_k$$

$$\hat{u}_i^* = \hat{u}_i \ \Rightarrow \ R^{*2} = R^2$$

(c) Using the above conclusions, we can get the result of demeaning: $y_i^* = y_i - \bar{y}$, $x_{ik}^* = x_{ik} - \bar{x}_k$, $k = \overline{1, K}$, $i = \overline{1, N}$. Now we will have:

$$\hat{\beta}_0^* = \hat{\beta}_0 + \sum_{k=1}^{K} \hat{\beta}_k \bar{x}_k - \bar{y} \qquad R^{*2} = R^2$$