PE I: Univariate Regression

Data subsampling & Chapter Review (Chapter 3)

Andrius Buteikis, andrius.buteikis@mif.vu.lt
http://web.vu.lt/mif/a.buteikis/

Splitting the data into training and validation sets

After estimating a model we may find it to be a good fit on out existing data. We then may be interested in predicting values based on some new unseen data.

While we have seen *how* to carry out these predictions, we have yet to consider *where* will we get the new data? Some possible sources are:

- Additional data that we get some time later;
- Creating some new data, which we think may be realistic, based on our existing observations, or specific scenario.
- Set aside part of the original data and exclude it from our model estimation. Then, use that data for prediction calculation.

While we have considered on the first two options - the last one may actually be more important - we can immediately examine how well our model performs in terms of predictions, based on a real subsample of the data.

The idea of setting aside part of the original dataset for model testing is known as the **holdout method** (since we *hold* part of the data and exclude it from parameter estimation).

The Holdout method

The idea is to **randomly** split the original data sample into **training** and **test** sets, where:

- The **training** set is a dataset used to fit the model parameters. Generally, the training set is set to be $\sim 80\%$ of the original data. The possible difficulties:
  - With less data in the training set, the parameter estimates have a greater variance;
  - With more data in the training set, the parameter estimates may be a better fit on the validation set, but a poor fit on new data.
- The **test** set is an *independent* dataset that is from the same **DGP** (**D**ata **G**enerating **P**rocess) as the validation set. It is usually selected to be the remaining $\sim 20\%$ of the original data (i.e. what is left after creating the training set).
- Sometimes an additional **validation** set is created if we want to adjust the estimated parameters (so for training/validation/test sets we take $70/15/15\%$ of the data). Then: we estimate the parameter via the *training* set, adjust the parameters via the *validation* set, and then use the **test** set to examine the predictive performance of our model. For now, we will focus on using training and test sets only.

The original data sample can also be **repeatedly randomly split** into **training** and **test** sets. This is known as **cross-validation**.

OLS: Assumptions

(UR.1) The Data Generating Process (**DGP**), or in other words, the population, is described by a linear (*in terms of the coefficients*) model:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i, \ \forall i = 1, ..., N \tag{UR.1}$$

(UR.2) The error term ϵ has an expected value of zero, given any value of the explanatory variable:

$$\mathbb{E}(\epsilon_i | X_j) = 0, \ \forall i, j = 1, ..., N \tag{UR.2}$$

(UR.3) The error term ϵ has the same variance given any value of the explanatory variable (i.e. homoskedasticity) and the error terms are not correlated across observations (i.e. no autocorrelation):

$$\mathbb{V}\mathrm{ar}\left(\boldsymbol{\varepsilon} | \mathbf{X}\right) = \sigma_\epsilon^2 \mathbf{I} \tag{UR.3}$$

i.e. $\mathbb{C}\mathrm{ov}(\epsilon_i, \epsilon_j) = 0, i \neq j$ and $\mathbb{V}\mathrm{ar}(\epsilon_i) = \sigma_\epsilon^2 = \sigma^2$ and the **identity** (or **unit**) **matrix** $\mathbf{I} = \mathbf{I}_N = \mathrm{diag}(1, ..., 1)$

(UR.4) (optional) The residuals are normal:

$$\boldsymbol{\varepsilon} | \mathbf{X} \sim \mathcal{N}\left(\mathbf{0}, \sigma_\epsilon^2 \mathbf{I}\right) \tag{UR.4}$$

$\boldsymbol{\varepsilon} = (\epsilon_1, ..., \epsilon_N)^\top$, $\mathbf{X} = (X_1, ..., X_N)^\top$, and $\mathbf{Y} = (Y_1, ..., Y_N)^\top$.

It is convenient to use matrices when solving equation systems. Looking at our random sample equations:

$$\begin{cases} Y_1 & = \beta_0 + \beta_1 X_1 + \epsilon_1 \\ Y_2 & = \beta_0 + \beta_1 X_2 + \epsilon_2 \\ \vdots & \\ Y_N & = \beta_0 + \beta_1 X_N + \epsilon_N \end{cases}$$

which we can re-write in the following matrix notation:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_N \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix}$$

Or, in a more compact form:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where $\mathbf{Y} = [Y_1, ..., Y_N]^\top$, $\boldsymbol{\varepsilon} = [\epsilon_1, ..., \epsilon_N]^\top$, $\boldsymbol{\beta} = [\beta_0, \beta_1]^\top$, $\mathbf{X} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_N \end{bmatrix}$.

OLS: The Estimator and Standard Errors

The unknown parameters of the linear regression $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ can be estimated via **OLS**:

$$\widehat{\boldsymbol{\beta}} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{Y} \qquad \text{(OLS)}$$

The term **Ordinary Least Squares (OLS)** comes from the fact that these estimates minimize the sum of squared residuals.

Gauss-Markov Theorem

Under the assumption that the conditions (UR.1) - (UR.3) hold true, the OLS estimator (OLS) is **BLUE** (**B**est **L**inear **U**nbiased **E**stimator) and **Consistent**.

The square roots of the diagonal elements of the variance-covariance matrix

$$\widehat{\mathbb{V}\mathrm{ar}}(\widehat{\boldsymbol{\beta}}) = \begin{bmatrix} \widehat{\mathbb{V}\mathrm{ar}}(\widehat{\beta}_0) & \widehat{\mathbb{C}\mathrm{ov}}(\widehat{\beta}_0, \widehat{\beta}_1) \\ \widehat{\mathbb{C}\mathrm{ov}}(\widehat{\beta}_1, \widehat{\beta}_0) & \widehat{\mathbb{V}\mathrm{ar}}(\widehat{\beta}_1) \end{bmatrix} = \widehat{\sigma}^2 \left(\mathbf{X}^\top \mathbf{X}\right)^{-1}, \text{ where } \widehat{\sigma}^2 = \frac{1}{N-2} \widehat{\boldsymbol{\varepsilon}}^\top \widehat{\boldsymbol{\varepsilon}},$$

are called **the standard errors (se)** of the corresponding (OLS) estimators $\widehat{\boldsymbol{\beta}}$, which we use to **estimate** the standard **deviation** of $\widehat{\beta}_i$ from β_i

$$\mathrm{se}(\widehat{\beta}_i) = \sqrt{\widehat{\mathbb{V}\mathrm{ar}}(\widehat{\beta}_i)}$$

The standard errors describe the accuracy of an estimator (the smaller the better).

Summary via Example

We want to create a model in order to determine how `educ` explains `wage`:

$$\text{wage} = f(\text{educ},\ \epsilon)$$

where $f(\cdot)$ is some **unknown** functional form, which is appropriate for our data.

The packages that we will need in Python are:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import scipy.stats as stats
#
from statsmodels.compat import lzip
import statsmodels.stats.diagnostic as sm_diagnostic
import statsmodels.stats.stattools as sm_tools
```

We will review the introduced methods and main Python functions by expanding on Task 24 from the lecture notes.

```
data_source = "http://www.principlesofeconometrics.com/poe5/data/csv/cps5_small.csv"
dt4 = pd.read_csv(data_source)
print(dt4.head(15))
```

```
##      black  educ  exper  faminc  female  metro  midwest  south   wage  west
## 0        0    13     45       0       1      1        0      0  44.44     1
## 1        0    14     25   45351       1      1        1      0  16.00     0
## 2        0    18     27   91946       1      1        0      0  15.38     0
## 3        0    13     42   48370       0      1        1      0  13.54     0
## 4        0    13     41   10000       1      1        0      0  25.00     1
## 5        0    16     26  151308       1      1        0      0  24.05     0
## 6        0    16     11  110461       1      1        0      1  40.95     0
## 7        0    18     15       0       1      1        1      0  26.45     0
## 8        0    21     32   67084       0      1        1      0  30.76     0
## 9        0    14     12   14000       0      0        0      0  34.57     1
## 10       0    18      8       0       1      1        0      1  20.67     0
## 11       0    12     40   25000       0      1        0      0  21.50     0
## 12       0    12     43       0       0      1        1      0  11.77     0
## 13       0    12     23       0       1      1        1      0  24.35     0
## 14       0    16     19   50808       0      1        0      1  55.00     0
```

We will subset the data as follows:

- We can get the index of each row as follows:

```
dt_index = list(range(0, len(dt4.index)))
print(dt_index[:5])
```

```
## [0, 1, 2, 3, 4]
```

- Then, 80% of the data will be equivalent to the rounded down number of observations:

```
print(int(np.floor(0.8 * len(dt4))))
```

```
## 960
```

- Subsetting the data is equivalent to taking a subset of the index vector

```
np.random.seed(123)
#
smpl_index = np.random.choice(dt_index, size = int(np.floor(0.8 * len(dt4))), replace = False)
print(smpl_index[:5])
```

```
## [156 920 971 897  35]
```

```
print(np.sort(smpl_index)[:5])
```

```
## [0 1 4 5 6]
```

Note that we are taking the index values **without replacement** - each element from the index can be taken only once.

- The remaining index values, which we do not take are:

```
print(list(set(dt_index) - set(smpl_index))[:5])
```

```
## [2, 3, 515, 516, 519]
```

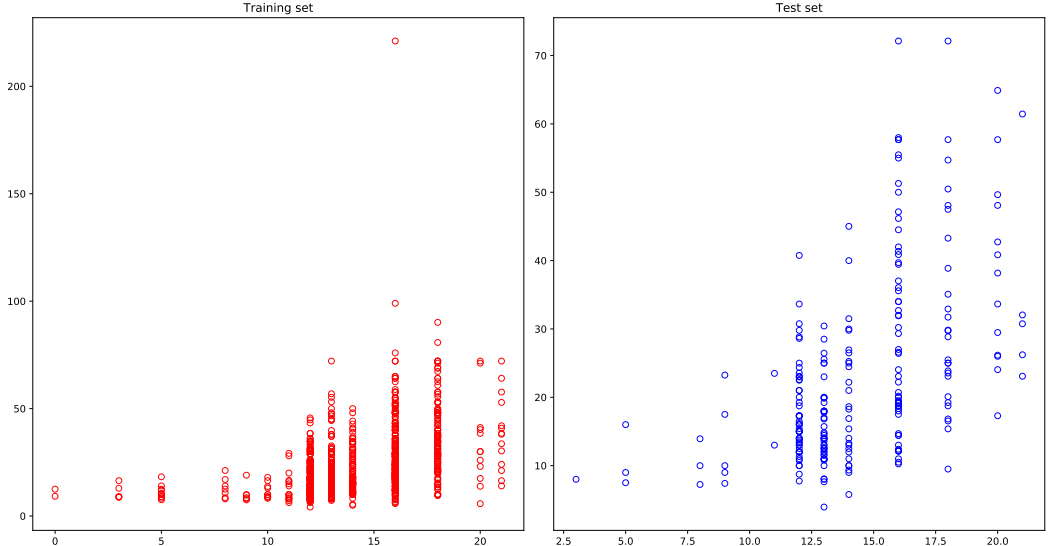
- To make it easier, we will create the **training** and **test** dataset from these indexes:

```
dt4_train = dt4.iloc[smpl_index, :].reset_index(drop = True)
dt4_test  = dt4.iloc[list(set(dt_index) - set(smpl_index)), ].reset_index(drop = True)
```

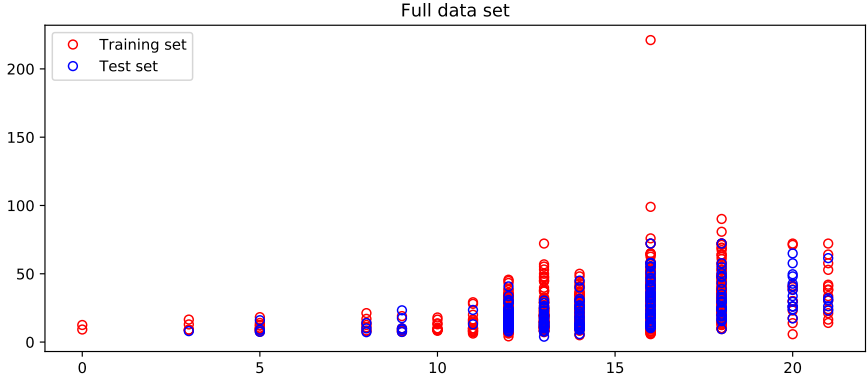
- We can plot the datasets as follows (to make it easier, we can define some default plot options via `plot_opts`):

```
plot_opts = dict(linestyle = "None", marker = "o", markerfacecolor = "None")
```

```
fig = plt.figure(num = 0, figsize = (15, 8))
_ = fig.add_subplot(121).plot(dt4_train[['educ']], dt4_train[['wage']],
          color = "red", **plot_opts)
_ = plt.title("Training set")
_ = fig.add_subplot(122).plot(dt4_test[['educ']], dt4_test[['wage']],
          color = "blue", **plot_opts)
_ = plt.title("Test set")
plt.tight_layout()
plt.show()
```


Training set
Test set
0
50
100
150
200
0
5
10
15
20
10
20
30
40
50
60
70
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

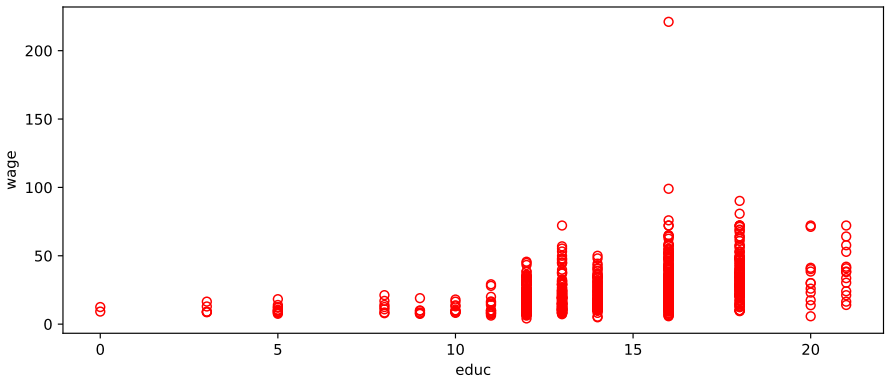
```
fig = plt.figure(num = 1, figsize = (10, 4))
_ = fig.add_subplot(111).plot(dt4_train[['educ']], dt4_train[['wage']],
        color = "red", label = "Training set", **plot_opts)
_ = fig.add_subplot(111).plot(dt4_test[['educ']], dt4_test[['wage']],
        color = "blue", label = "Test set", **plot_opts)
_ = plt.title("Full data set")
_ = plt.legend(loc = "upper left")
plt.show()
```



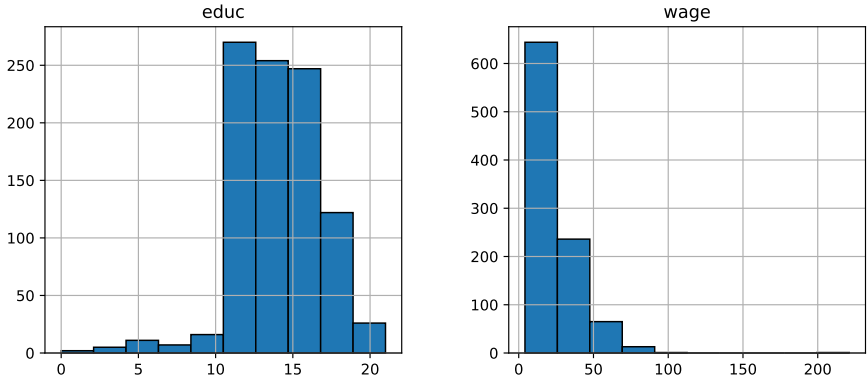
From this point on, we will use the `train` set for model estimation and the `test` set for prediction.

Examining The Data

```
fig = plt.figure(num = 2, figsize = (10, 4))
_ = plt.plot(dt4_train[['educ']], dt4_train[['wage']], color = "red", **plot_opts)
_ = plt.xlabel("educ")
_ = plt.ylabel("wage")
plt.show()
```



```
fig = plt.figure(num = 2, figsize = (10, 4))
_ = dt4_train[['educ']].hist(ec = 'black', ax = fig.add_subplot(121))
_ = dt4_train[['wage']].hist(ec = 'black', ax = fig.add_subplot(122))
plt.show()
```



```
print(dt4_train[['educ', 'wage']].describe())
```

```
##              educ        wage
## count  960.000000  960.000000
## mean    14.151042   23.752979
## std      2.852491   15.638523
## min      0.000000    4.170000
## 25%     12.000000   12.995000
## 50%     14.000000   19.590000
## 75%     16.000000   29.850000
## max     21.000000  221.100000
```

```
print(pd.DataFrame([{">=10": np.sum(dt4_train['educ'] >= 10),
                     "<10": np.sum(dt4_train['educ'] < 10)}]))
```

```
##    >=10  <10
## 0   929   31
```

We see that:

- The minimum value of `educ` is 0 - this **immediately** rules out `linear-log` and `log-log` models;
- Higher `educ` appears to lead to larder values of `wage`, but the variation in `wage` also increases.
- It appears that we have more observations for $educ \geq 10$, than when $educ < 10$. This will most likely result in a less accurate model for $educ < 10$.
- Noting the fact that if (UR.4) holds, then it should also hold that $\boldsymbol{Y}|\boldsymbol{X} \sim \mathcal{N}(\cdot, \cdot)$, we see that `wage` does not appear to be normally distributed. It is likely that the residuals are also not normally distributed, if we were to fit a **simple linear regression model**.

We expect that the more years one spends in `education`, the more qualified they are for higher position (i.e. higher-paying) jobs. As such, the hourly `wage` should increase with more years spent on studying. So, we expect $\beta_1 > 0$.

Model Estimation

We can estimate three models:

- The **level-level** model: $wage = \beta_0 + \beta_1 educ + \epsilon$:

```
lm_fit_linear = sm.OLS(dt4_train['wage'], sm.add_constant(dt4_train['educ'])).fit()
print(lm_fit_linear.summary2().tables[1].round(4))
```

```
##            Coef.  Std.Err.         t  P>|t|    [0.025  0.975]
## const -10.4726    2.2947  -4.5639    0.0 -14.9757 -5.9695
## educ    2.4186    0.1590  15.2150    0.0   2.1066  2.7305
```

- The **quadratic** model: $wage = \beta_0 + \beta_1 educ^2 + \epsilon$:

```
lm_fit_quad = sm.OLS(dt4_train['wage'], sm.add_constant(dt4_train['educ']**2)).fit()
print(lm_fit_quad.summary2().tables[1].round(4))
```

```
##          Coef.  Std.Err.         t   P>|t|   [0.025  0.975]
## const  4.7238    1.2825    3.6833  0.0002   2.207  7.2406
## educ   0.0913    0.0058   15.8427  0.0000   0.080  0.1026
```

- The **log-level** model: $\log(wage) = \beta_0 + \beta_1 educ + \epsilon$:

```
lm_fit_loglin = sm.OLS(np.log(dt4_train['wage']), sm.add_constant(dt4_train['educ'])).fit()
print(lm_fit_loglin.summary2().tables[1].round(4))
```

```
##          Coef.  Std.Err.         t  P>|t|   [0.025  0.975]
## const  1.6018    0.0807  19.8546    0.0  1.4435  1.7601
## educ   0.0988    0.0056  17.6812    0.0  0.0878  0.1098
```

Coefficient significance test

We can test the hypothesis:

$$\begin{cases} H_0 & : \beta_1 = 0 \\ H_1 & : \beta_1 \neq 0 \end{cases}$$

From the model output `P>|t|` column we see that the `p-value < 0.05` - so we reject the null hypothesis and conclude that `educ` has a **significant** effect on `wage` in all three of our estimated models.

We also see that $\beta_1 > 0$ in all of the models, which coincides with what we expect - `educ` has a positive effect on `wage`. The exact interpretation depends on the model specification...

Model Interpretation

We can write down and interpret the three models as follows:

- $\underset{(se)}{\widehat{wage}} = \underset{(2.2947)}{-10.4726} + \underset{(0.1590)}{2.4186} \cdot educ$:
  - An additional **year** in `educ` results in a 2.4186 **dollar increase** in `wage`.

- $\underset{(se)}{\widehat{wage}} = \underset{(1.2825)}{4.7238} + \underset{(0.0058)}{0.0913} \cdot educ^2$:
  - An additional **year** in `educ` results in an **increase** in `wage`. Furthermore, the increase is more **pronounced** for larger values of `educ`.
  - Note that the increase can be calculated as a $4.7238 + 0.0913 \cdot (educ + 1)^2 - (4.7238 + 0.0913 \cdot educ^2) = 0.0913 \cdot (2 \cdot educ + 1)$ **dollar increase** in `wage`.

- $\underset{(se)}{\widehat{\log(wage)}} = \underset{(0.0807)}{1.6018} + \underset{(0.0056)}{0.0988} \cdot educ$:
  - An additional **year** in `educ` results in a $0.0988 \cdot 100 = 9.88$ **percent increase** in `wage`.

Plotting the Fitted values

We will begin by saving the index order of the **sorted** values of X (in this case, `educ`) for plotting:

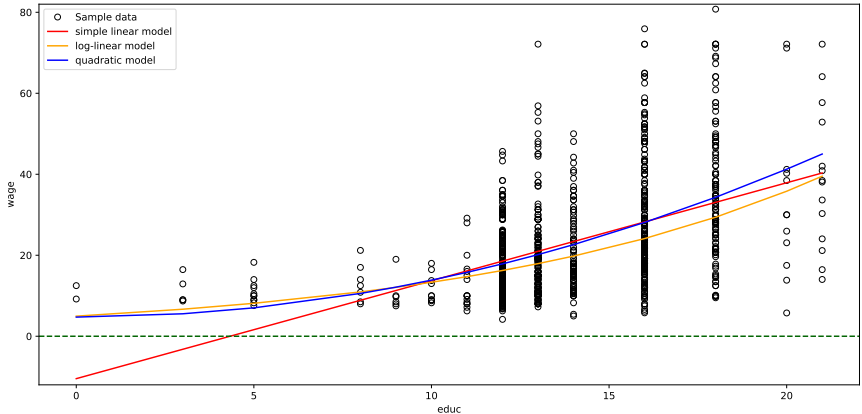
```
sorted_vals = np.argsort(dt4_train['educ'])
```

We then calculate the *fitted* values of Y (in this case, `wage`):

```
y_fit_linear = lm_fit_linear.fittedvalues
y_fit_quad = lm_fit_quad.fittedvalues
y_fit_loglin = np.exp(lm_fit_loglin.fittedvalues)
```

We can then plot the fitted values from the different models to visually examine them:

```
fig = plt.figure(num = 3, figsize = (15, 7))
_ = plt.plot(dt4_train['educ'], dt4_train['wage'], label = "Sample data",
             color = "black", **plot_opts)
_ = plt.plot(dt4_train['educ'][sorted_vals], y_fit_linear[sorted_vals],
             linestyle = "-", color = "red", label = "simple linear model")
_ = plt.plot(dt4_train['educ'][sorted_vals], y_fit_loglin[sorted_vals],
             linestyle = "-", color = "orange", label = "log-linear model")
_ = plt.plot(dt4_train['educ'][sorted_vals], y_fit_quad[sorted_vals],
             linestyle = "-", color = "blue", label = "quadratic model")
_ = plt.axhline(y = 0, linestyle = "--", color = "darkgreen")
_ = plt.xlabel("educ")
_ = plt.ylabel("wage")
_ = plt.ylim(bottom = -12, top = 82)
_ = plt.legend(loc = "upper left")
plt.show()
```



The linear model provides a poor fit for *educ* < 5 - it is unrealistic that `wage` < 0.

From this alone we have a strong indication to rule-out the simple linear (i.e. **level-level**) model, since the low values of `educ` come from the data sample itself, and therefore are realistic.

Residual Diagnostic Plots

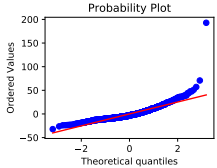
```
def plot_resid(resid, y_fit, x, plt_title, plt_row, plt_pos, fig):
    plot_opts = dict(linestyle = "None", marker = "o", markerfacecolor = "None")
    # Plot Text Box with `plt_title`
    ax = fig.add_subplot(plt_row, 5, plt_pos)
    ax.set_yticklabels([])
    ax.set_xticklabels([])
    ax.tick_params(right = False, top = False, left = False, bottom= False)
    ax.text(0.5, 0.5, plt_title, horizontalalignment='center',
            verticalalignment='center', transform = ax.transAxes)
    # Plot Q-Q plot of residuals
    ax = fig.add_subplot(plt_row, 5, plt_pos + 1)
    stats.probplot(resid, dist = "norm", plot = ax)
    # Plot histogram of residuals
    ax = fig.add_subplot(plt_row, 5, plt_pos + 2)
    ax.hist(resid, color = "cornflowerblue", bins = 25, ec = 'black')
    plt.title("Histogram of residuals")
    # Plot residual vs Fitted plot
    ax = fig.add_subplot(plt_row, 5, plt_pos + 3)
    ax.plot(y_fit, resid, color = "black", **plot_opts)
    plt.title("Residuals vs Fitted values")
    # Plot residual vs X plot
    ax = fig.add_subplot(plt_row, 5, plt_pos + 4)
    ax.plot(x, resid, color = "black", **plot_opts)
    plt.title("Residuals vs X")
#
```

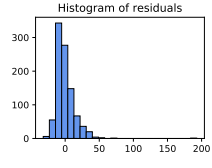
```
fig = plt.figure(num = 4, figsize = (16, 8))
#
plot_resid(lm_fit_linear.resid, lm_fit_linear.fittedvalues, dt4_train['educ'],
           "simple linear model", 3, 1, fig)
plot_resid(lm_fit_quad.resid, lm_fit_quad.fittedvalues, dt4_train['educ'],
           "linear-log model", 3, 5 * 1 + 1, fig)
plot_resid(lm_fit_loglin.resid, lm_fit_loglin.fittedvalues, dt4_train['educ'],
           "log-linear model", 3, 5 * 2 + 1, fig)
#
plt.tight_layout()
plt.show()
```

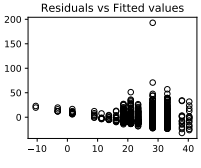
(See next slide for the plots)

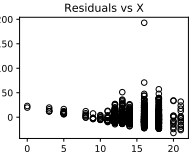
- For the **simple linear model** - it appears that the residuals are skewed and from a non-normal distribution - the residuals on the Q-Q plot do not fall along a straight line with the theoretical quantiles of a normal distribution. Furthermore, the histogram indicates that the residual distribution is skewed to the right hand side of the histogram.
- For the **quadratic model** - the residuals of a quadratic model do not appear to be from a normal distribution - they are quite similar to the residuals from a simple linear model.
- For the **log-linear model** - the residuals of a log-linear model appear to closely resemble a normal distribution - the histogram resembles a bell shape that is familiar of a normal distribution, while the Q-Q plot falls along a straight line.

simple linear model

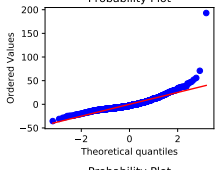


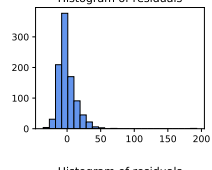


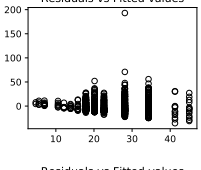


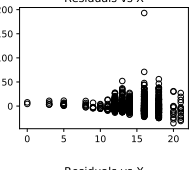


quadratic model

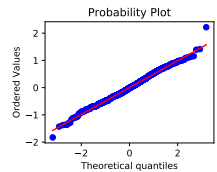


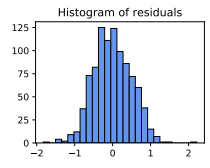


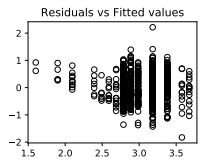


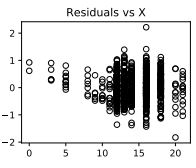


log-linear model









Residual Diagnostic Tests

We will carry out a number of tests on the residuals and test:

- Whether the residuals are homoskedastic
  (if they are not - then $\exists i \in \{1, .., n\} : \ \mathbb{V}\mathrm{ar}(\epsilon_i) \neq \sigma^2$ and (UR.3) is violated)
- Whether the residuals are uncorrelated
  (if they are not - then $\exists i, j \in \{1, .., n\} : \ \mathbb{C}\mathrm{ov}(\epsilon_i, \epsilon_j) \neq 0, \ i \neq j$ and (UR.3) is violated)
- Whether the residuals are normally distributed
  (if they are not - then $\boldsymbol{\varepsilon} \not\sim \mathcal{N}(\cdot, \cdot)$ and (UR.4) is violated)

Testing for Residual Homoskedasticity

We want to test the hypothesis:

$$\begin{cases} H_0 & : \text{residuals are homoskedastic} \\ H_1 & : \text{residuals are heteroskedastic} \end{cases}$$

```
name = ['LM statistic', 'LM p-value', 'F-value', 'F  p-value']
```

- For the **level-level model**:

```
# Breusch-Pagan test
test = sm_diagnostic.het_breuschpagan(
       resid = lm_fit_linear.resid, exog_het = sm.add_constant(dt4_train[["educ"]]))
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    LM statistic  LM p-value   F-value  F  p-value
## 0      5.692839    0.017034  5.714869    0.017014
```

- For the **log-linear model**:

```
# Breusch-Pagan test
test = sm_diagnostic.het_breuschpagan(
       resid = lm_fit_quad.resid, exog_het = sm.add_constant(dt4_train[["educ"]]))
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    LM statistic  LM p-value   F-value  F  p-value
## 0      7.238903    0.007134  7.278708      0.0071
```

- For the **log-linear model**:

```
# Breusch-Pagan test
test = sm_diagnostic.het_breuschpagan(
       resid = lm_fit_loglin.resid, exog_het = sm.add_constant(dt4_train[["educ"]]))
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    LM statistic  LM p-value   F-value  F  p-value
## 0      5.706901    0.016898  5.729069    0.016878
```

Testing for Residual Autocorrelation

We want to test the hypothesis:

$$\begin{cases} H_0 & : \text{residuals are serially uncorrelated} \\ H_1 & : \text{residuals are autocorrelated} \end{cases}$$

```
name = ['DW statistic']
```

- For the **level-level model**:

```
# Durbin-Watson Test
test = sm_tools.durbin_watson(lm_fit_linear.resid)
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    DW statistic
## 0      2.028807
```

- For the **log-linear model**:

```
# Durbin-Watson Test
test = sm_tools.durbin_watson(lm_fit_quad.resid)
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    DW statistic
## 0      2.018592
```

- For the **log-linear model**:

```
# Durbin-Watson Test
test = sm_tools.durbin_watson(lm_fit_loglin.resid)
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    DW statistic
## 0        2.0461
```

Testing for Residual Normality

We want to test the hypothesis:

$$\begin{cases} H_0 & : \text{residuals follow a normal distribution} \\ H_1 & : \text{residuals do not follow a normal distribution} \end{cases}$$

```
name = ['W-statistic', 'p-value']
```

- For the **level-level model**:

```
# Shapiro-Wilk Test
test = stats.shapiro(x = lm_fit_linear.resid)
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    W-statistic  p-value
## 0     0.810388      0.0
```

- For the **log-linear model**:

```
# Shapiro-Wilk Test
test = stats.shapiro(x = lm_fit_quad.resid)
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    W-statistic  p-value
## 0     0.812549      0.0
```

- For the **log-linear model**:

```
# Shapiro-Wilk Test
test = stats.shapiro(x = lm_fit_loglin.resid)
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    W-statistic   p-value
## 0     0.995557  0.007039
```

```
name = ['JB statistic', 'p-value', 'skewness', 'kurtosis']
```

- For the **level-level model**:

```
# Jarque-Bera Test
test = sm_tools.jarque_bera(lm_fit_linear.resid)
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    JB statistic  p-value  skewness  kurtosis
## 0   58839.95575      0.0  3.579048  40.67971
```

- For the **log-linear model**:

```
# Jarque-Bera Test
test = sm_tools.jarque_bera(lm_fit_quad.resid)
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    JB statistic  p-value  skewness   kurtosis
## 0  63168.042352      0.0  3.611011  42.077403
```

- For the **log-linear model**:

```
# Jarque-Bera Test
test = sm_tools.jarque_bera(lm_fit_loglin.resid)
print(pd.DataFrame([np.round(test, 6)], columns = name))
```

```
##    JB statistic   p-value  skewness  kurtosis
## 0      1.604277  0.448369  0.086493  3.100911
```

Residual Diagnostic Tests: Conclusions

We can conclude that:

- From the **Breusch-Pagan** test - the p-value < 0.05, so **in each of our models** we reject the null hypothesis of residual homoskedasticity and conclude that the residuals are heteroskedastic (**this is bad**);
- From the **Durbin-Watson** test - the DW statistic is close to 2 for all models, so **in each of our models** we conclude that the residuals **not** correlated (**this is good**);
- From the **Shapiro-Wilk** test - the p-value < 0.05, so **in each of our models** we reject the null hypothesis of residual normality and conclude that the residuals are not normally distributed (**this is bad**);
- From the **Jarque-Bera** test - the p-value > 0.05 for the **log-linear model**, so we have no grounds to reject the null hypothesis that the residuals of the log-linear model are normally distributed (**this is good**).

What does this say about our coefficient estimates in these models? What about their test statistics and confidence intervals? See the lecture notes.

Only the **Jarque-Bera** test for normality gives a different (and a favorable) conclusion for the **log-linear** model. As such, we can conclude that the **log-linear** model is the best *out of the three models*, which we have considered. However, the residuals of this model are still heteroskedastic.

The R^2 of the **log-linear** model:

```
print(lm_fit_loglin.rsquared)
```

```
## 0.24604096498074746
```

Indicates that around 24.6% of the variation in log(*wage*) is explained by (the changes in) *educ* in our **log-linear** model.

Alternatively:

```
print(np.corrcoef(dt4_train['wage'], np.exp(lm_fit_loglin.fittedvalues))[0][1]**2)
```

```
## 0.2046178787933547
```

Around 20.46 of the variance in *wage* is explained by (the changes in) *educ* in our **log-linear** model.

Fitted Values, Confidence and Prediction intervals: Training set

```
sorted_train = np.argsort(dt4_train['educ'])
```

We will plot the fitted values, the confidence intervals of the mean response and the prediction intervals for the **training** set - that is, the data, which we have used to estimate our parameters:

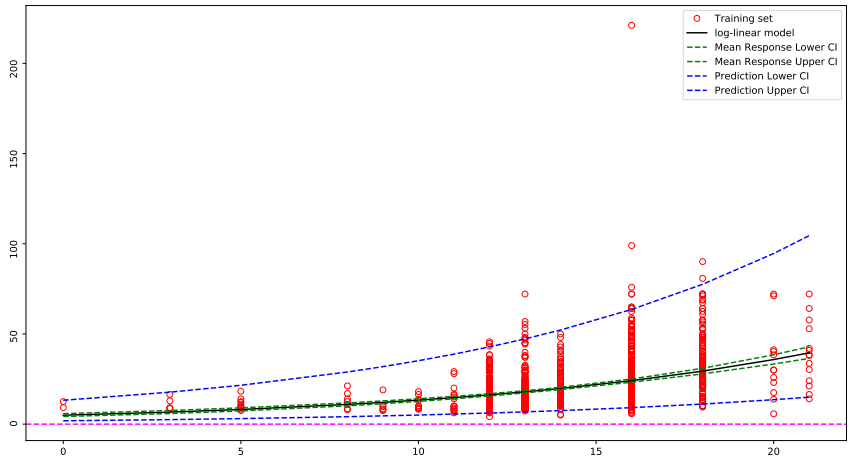
```
fit_train = lm_fit_loglin.get_prediction(sm.add_constant(dt4_train["educ"]))
fit_train = fit_train.summary_frame(alpha = 0.05)
print(fit_train.head(5))
```

```
##        mean   mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  obs_ci_upper
## 0  2.787617  0.019960       2.748446       2.826788      1.817986      3.757248
## 1  2.886435  0.017183       2.852713       2.920156      1.917009      3.855861
## 2  2.787617  0.019960       2.748446       2.826788      1.817986      3.757248
## 3  2.787617  0.019960       2.748446       2.826788      1.817986      3.757248
## 4  3.380522  0.026770       3.327988       3.433056      2.410259      4.350784
```

```
print(np.exp(fit_train).head(5))
```

```
##         mean   mean_se  ...  obs_ci_lower  obs_ci_upper
## 0  16.242271  1.020161  ...      6.159443     42.830394
## 1  17.929271  1.017332  ...      6.800585     47.269279
## 2  16.242271  1.020161  ...      6.159443     42.830394
## 3  16.242271  1.020161  ...      6.159443     42.830394
## 4  29.386103  1.027131  ...     11.136849     77.539262
##
## [5 rows x 6 columns]
```

```
fig = plt.figure(num = 5, figsize = (15, 8))
_ = plt.plot(dt4_train[['educ']], dt4_train[['wage']], color = "red", **plot_opts,
             label = "Training set")
_ = plt.plot(dt4_train['educ'][sorted_train], np.exp(fit_train)['mean'][sorted_train],
             linestyle = "-", color = "black", label = "log-linear model")
_ = plt.plot(dt4_train['educ'][sorted_train], np.exp(fit_train)['mean_ci_lower'][sorted_train],
             linestyle = "--", color = "green", label = "Mean Response Lower CI")
_ = plt.plot(dt4_train['educ'][sorted_train], np.exp(fit_train)['mean_ci_upper'][sorted_train],
             linestyle = "--", color = "green", label = "Mean Response Upper CI")
_ = plt.plot(dt4_train['educ'][sorted_train], np.exp(fit_train)['obs_ci_lower'][sorted_train],
             linestyle = "--", color = "blue", label = "Prediction Lower CI")
_ = plt.plot(dt4_train['educ'][sorted_train], np.exp(fit_train)['obs_ci_upper'][sorted_train],
             linestyle = "--", color = "blue", label = "Prediction Upper CI")
_ = plt.axhline(y = 0, linestyle = "--", color = "magenta")
_ = plt.yticks(rotation = 'vertical')
_ = plt.legend(loc = "upper right")
plt.show()
```


Training set
log-linear model
Mean Response Lower CI
Mean Response Upper CI
Prediction Lower CI
Prediction Upper CI
200
150
100
50
0
0
5
10
15
20

Fitted Values, Confidence and Prediction intervals: Test set

Now, we will use completely new data and examine the model predictions:

```
sorted_test = np.argsort(dt4_test['educ'])
```

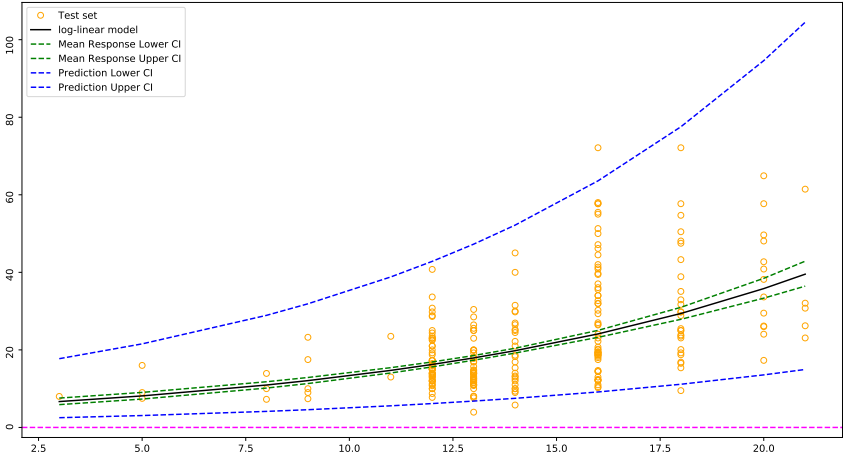
```
fit_test = lm_fit_loglin.get_prediction(sm.add_constant(dt4_test["educ"]))
fit_test = fit_test.summary_frame(alpha = 0.05)
print(fit_test.head(5))
```

```
##        mean   mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  obs_ci_upper
## 0  3.380522  0.026770       3.327988       3.433056      2.410259      4.350784
## 1  2.886435  0.017183       2.852713       2.920156      1.917009      3.855861
## 2  2.886435  0.017183       2.852713       2.920156      1.917009      3.855861
## 3  3.380522  0.026770       3.327988       3.433056      2.410259      4.350784
## 4  3.182887  0.018991       3.145618       3.220156      2.213331      4.152443
```

```
print(np.exp(fit_test).head(5))
```

```
##         mean    mean_se  ...  obs_ci_lower  obs_ci_upper
## 0  29.386103  1.027131  ...     11.136849     77.539262
## 1  17.929271  1.017332  ...      6.800585     47.269279
## 2  17.929271  1.017332  ...      6.800585     47.269279
## 3  29.386103  1.027131  ...     11.136849     77.539262
## 4  24.116276  1.019173  ...      9.146133     63.589148
##
## [5 rows x 6 columns]
```

```
fig = plt.figure(num = 6, figsize = (15, 8))
_ = plt.plot(dt4_test[['educ']], dt4_test[['wage']], color = "orange", **plot_opts,
             label = "Test set")
_ = plt.plot(dt4_test['educ'][sorted_test], np.exp(fit_test)['mean'][sorted_test],
             linestyle = "-", color = "black", label = "log-linear model")
_ = plt.plot(dt4_test['educ'][sorted_test], np.exp(fit_test)['mean_ci_lower'][sorted_test],
             linestyle = "--", color = "green", label = "Mean Response Lower CI")
_ = plt.plot(dt4_test['educ'][sorted_test], np.exp(fit_test)['mean_ci_upper'][sorted_test],
             linestyle = "--", color = "green", label = "Mean Response Upper CI")
_ = plt.plot(dt4_test['educ'][sorted_test], np.exp(fit_test)['obs_ci_lower'][sorted_test],
             linestyle = "--", color = "blue", label = "Prediction Lower CI")
_ = plt.plot(dt4_test['educ'][sorted_test], np.exp(fit_test)['obs_ci_upper'][sorted_test],
             linestyle = "--", color = "blue", label = "Prediction Upper CI")
_ = plt.axhline(y = 0, linestyle = "--", color = "magenta")
_ = plt.yticks(rotation = 'vertical')
_ = plt.legend(loc = "upper left")
plt.show()
```



We see that the prediction intervals in the **log-linear** model contain most (but not all) of the true observations from the test set. We also note that in this case, the test set did not contain the outliers, which were present in the training set. Furthermore, both the fitted values and the prediction intervals are positive, which is inline with our expectations, since we are modelling *wage*.