PE I: Univariate Regression

Hypothesis Testing (Review), OLS Prediction and Prediction Intervals, GoF (Chapters 3.6, 3.7 & 3.8)

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

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):

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

i.e. $\mathbb{C}\text{ov}(\epsilon_i, \epsilon_j) = 0, i \neq j$ and $\mathbb{V}\text{ar}(\epsilon_i) = \sigma_\epsilon^2 = \sigma^2$.

(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 $\mathbf{Y} = (Y_1, ..., Y_N)^\top$.

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).

Parameter Interval Estimators

- **Point estimates**, $\widehat{\beta}$, are *single values* for each parameter via (OLS).
- **Interval estimates** are *ranges of values*, in which the **true** β parameters are likely to fall. These interval estimates are also known as **confidence intervals**.

The **interval estimator** of β_i, $i = 0, 1$ is defined by these endpoints:

$$\widehat{\beta}_i \pm t_c \cdot \text{se}(\widehat{\beta}_i)$$

where t_c is the **critical value**, such that for a selected **significance level** α:

$$\mathbb{P}\left(\widehat{\beta}_i - t_c \cdot \text{se}(\widehat{\beta}_i) \leq \beta_i \leq \widehat{\beta}_i + t_c \cdot \text{se}(\widehat{\beta}_i)\right) = 1 - \alpha$$

For a specific sample, this is also known as a $100 \cdot (1 - \alpha)\%$ **interval estimate of** β_i, or the $100 \cdot (1 - \alpha)\%$ **confidence interval**. Here, we can also define the **t-statistic**:

$$t_i = \frac{\widehat{\beta}_i - \beta_i}{\text{se}(\widehat{\beta}_i)} \sim t_{(N-2)}, \quad \text{[which also means that } \mathbb{P}(-t_c \leq t_i \leq t_c) = 1 - \alpha\text{]}$$

also known as the **t-ratio**, which has a $t_{(N-2)}$ distribution under $H_0 : \widehat{\beta}_i = \beta_i$.

Question: What is a test statistic, t_i?

Answer: A test statistic is computed from the data under a specific **null hypothesis** and tested against pre-determined *critical values* for a selected *significance level*.

Question: What is the significance level, α?

Answer: α can be though of as the probability of making a **Type I** error - i.e. that we will reject the null hypothesis $(100 \cdot \alpha)\%$ of the time, when it is in fact **true**. They define the sensitivity of the test.

The choice of α is somewhat arbitrary, although in practice the values are usually selected quite small: $\alpha \in \{0.01,\ 0.05,\ 0.1\}$.

Question: What is the critical value, t_c?

Answer: Critical values are cut-off values that define regions, where the test statistic is **unlikely to lie**. The null hypothesis is **rejected** if the test statistic lies within this region (also known as the **rejection region**), which is an unlikely event to occur under the null.

In general, for every hypothesis test there is an equivalent statement about whether the hypothesized parameter value is included in a confidence interval.

Interval Estimation for the Mean Response

If (UR.4) assumption holds true, then, for some value of $\mathbf{X} = \widetilde{\mathbf{X}}$, the mean response $\widehat{\mathbf{Y}} = \widehat{\mathbb{E}}(\mathbf{Y}|\mathbf{X} = \widetilde{\mathbf{X}}) = \widetilde{\mathbf{X}}\widehat{\beta}$ follows a normal distribution:

$$\left(\widehat{\mathbf{Y}}|\widetilde{\mathbf{X}}, \mathbf{X}\right) \sim \mathcal{N}\left(\widetilde{\mathbf{X}}\beta, \quad \sigma^2\widetilde{\mathbf{X}}\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\widetilde{\mathbf{X}}^\top\right)$$

and the $100 \cdot (1-\alpha)\%$ **confidence interval** for the mean response is:

$$\widehat{Y}_i \pm t_{(1-\alpha/2, N-2)} \cdot \mathrm{se}(\widehat{Y}_i)$$

Furthermore, note that the *t*-distribution approaches the standard normal distribution as the degrees of freedom increases (i.e. as the sample size N gets larger).

In particular for $\alpha = 0.05$ we have that $t_{(1-\alpha/2,\ N)} \to 1.96$ as $N \to \infty$.

Hence, **a rule of thumb** for calculating *approximate* confidence intervals is to take $t_c \approx 1.96$ for $\alpha = 0.05$, if we have no way of calculating the actual value of $t_{(1-\alpha/2,\ N)}$.

The p-value approach to Hypothesis testing

The p-value is the **probability** that the test-statistic exceeds or equals the value of the test-statistic derived from a randomly drawn sample, **under** H_0.

The p-value **is not** the probability of a **Type I** error - it is given by a chosen value of α.

- If you collect data **many times**, under the assumption that **the null is true**, a proportion of α of those times you would reject the null (i.e. a **Type I** error);
- The p-value is a random variable computed from the actual observed data that can be compared to α as one way of performing the test (instead of comparing the test-statistics). See Murdock, D, Tsai, Y, and Adcock, J (2008) P-Values are Random Variables.

Instead of comparing the test-statistics to the rejection region, we can instead calculate the **probability** of what we observe (or more extreme) given the null is true and compare that probability to α. So, having the p-value allows us to easier determine the outcome of the test, as we do not need to directly compare the critical values.

- If p-value $< \alpha$, we **reject** H_0.
- If p-value $\geq \alpha$, we **do not reject** H_0.

Note: in practice, if p-value $\approx \alpha$, we may be indecisive on whether we should reject the null hypothesis, or not. After all, the p-value is based on a specific sample, so having a bit more data may result in a smaller/larger p-value.

Hypothesis Testing

Alternative, $>$ (One Tail Test)

$$\begin{cases} H_0 & : \beta_i = c \\ H_1 & : \beta_i > c \end{cases}$$

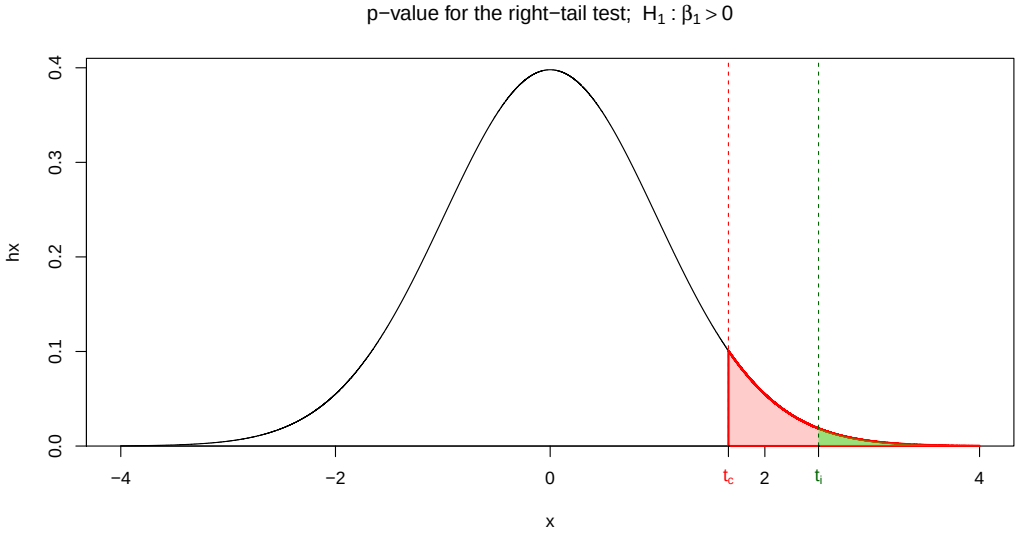
We **reject** H_0 and **accept the alternative** H_1, if $t_i \geq t_c$, where $t_c = t_{(1-\alpha, N-2)}$.

The p-value is the probability to the right of the calculated t-statistic:

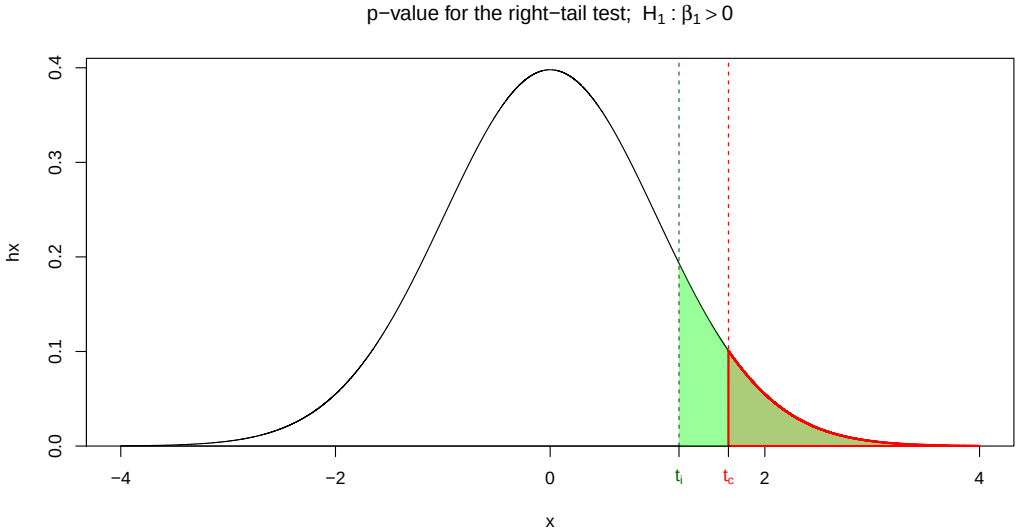
$$p\text{-value} = \mathbb{P}(T \geq t_i) = \mathbb{P}(T > t_i) = 1 - \mathbb{P}(T \leq t_i) = 1 - \int_{-\infty}^{t_i} p(x)dx$$

- If p-value $< \alpha$, we **reject** H_0.
- If p-value $\geq \alpha$, we **do not reject** H_0.

Here $p(x)$ is the density function of the distribution of t-statistic **under the null hypothesis**.



We reject H_0. From the shaded plots it is clear that p-value $< \alpha$.



We have no grounds to reject H_0. From the shaded plots it is clear that p-value $> \alpha$.

Hypothesis Testing

Alternative, $<$ (One Tail Test)

$$\begin{cases} H_0 & : \beta_i = c \\ H_1 & : \beta_i < c \end{cases}$$

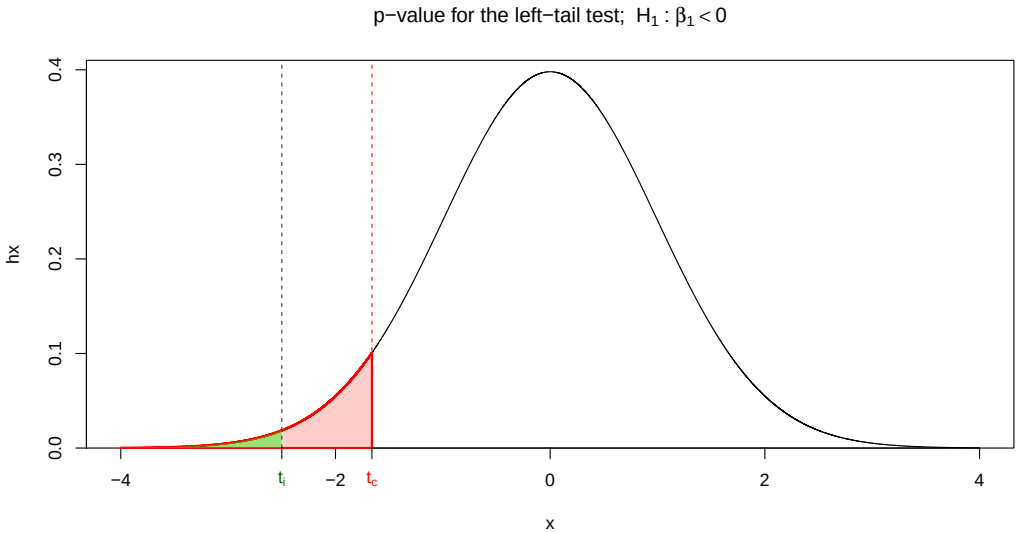
We **reject** H_0 and **accept the alternative** H_1, if $t_i \leq t_c$, where $t_c = t_{(\alpha, N-2)}$.

The p-value is the probability to the left of the calculated t-statistic:

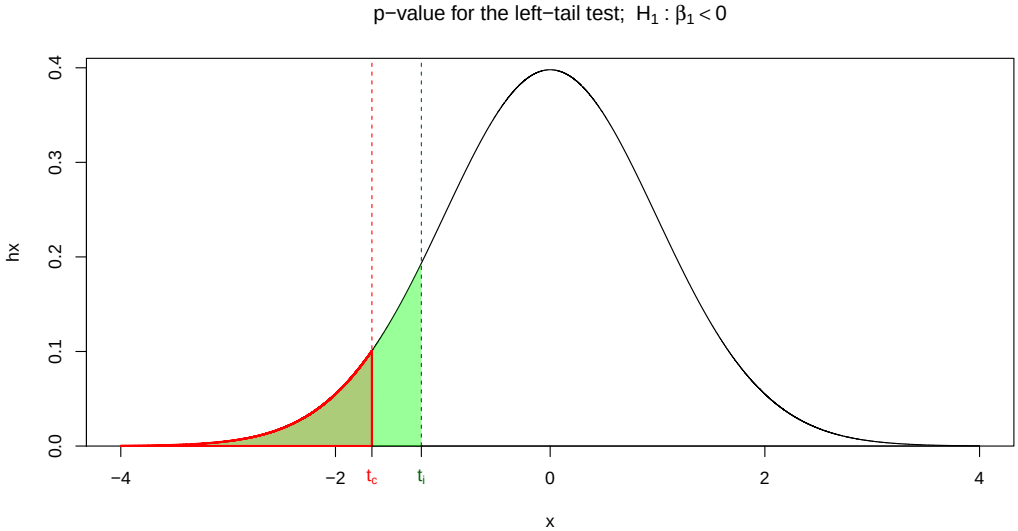
$$p\text{-value} = \mathbb{P}(T \leq t_i) = \int_{-\infty}^{t_i} p(x)dx$$

- If p-value $< \alpha$, we **reject** H_0.
- If p-value $\geq \alpha$, we **do not reject** H_0.

Here $p(x)$ is the density function of the distribution of t-statistic **under the null hypothesis**.



We reject H_0. From the shaded plots it is clear that p-value $< \alpha$.



We have no grounds to reject H_0. From the shaded plots it is clear that p-value $> \alpha$.

Hypothesis Testing

Alternative, $\neq$ (Two Tail Test)

$$\begin{cases} H_0 & : \beta_i = c \\ H_1 & : \beta_i \neq c \end{cases}$$

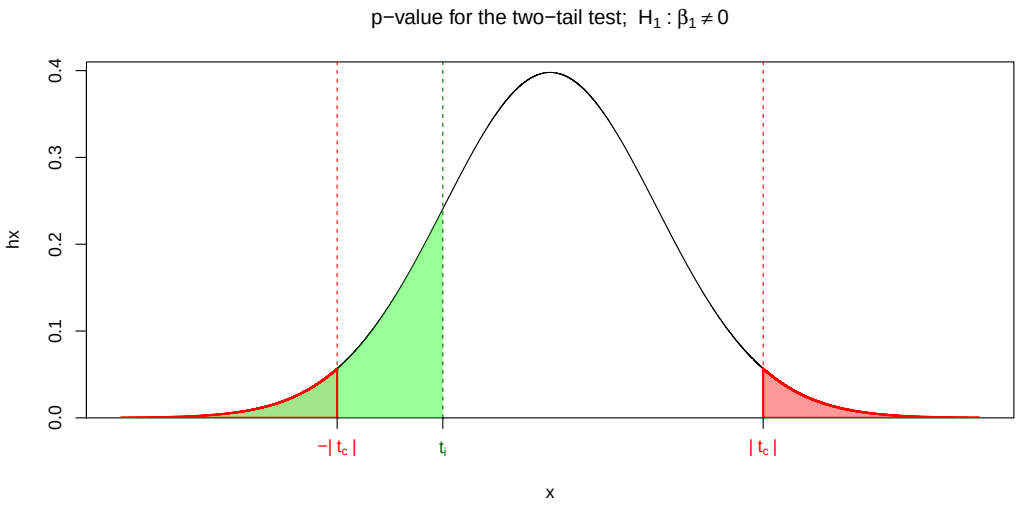
We **reject** H_0 and **accept the alternative** H_1, if $t_i \leq -|t_c|$ **or** if $t_i \geq |t_c|$, where $t_c = t_{(1-\alpha/2, N-2)} = -t_{(\alpha/2, N-2)}$.

The p-value is the probability to the left of the calculated t-statistic:

$$p\text{-value} = \mathbb{P}(T \leq -|t_i|) + \mathbb{P}(T \geq |t_i|) = 2 \cdot \mathbb{P}(T \leq -|t_i|) = 2 \cdot \int_{-\infty}^{-|t_i|} p(x)dx$$

- If p-value $< \alpha$, we **reject** H_0.
- If p-value $\geq \alpha$, we **do not reject** H_0.

Here $p(x)$ is the density function of the distribution of t-statistic **under the null hypothesis**.



We have no grounds to reject H_0. Note that in this case, we are checking either $t_i \leq -|t_c|$, **or** $t_i \geq |t_c|$, so here we shade an appropriate comparison, since the p-value itself is 2 times the probability to the left of $-|t_c|$.

Hypothesis Testing and The Parameter Confidence Interval

The two-tail tests and parameter confidence intervals are closely related. Assume that we want to test the following:

$$\begin{cases} H_0 & : \beta_i = c \\ H_1 & : \beta_i \neq c \end{cases}$$

Under the null hypothesis we would have that $\beta_i = c$. So, if we test the null hypothesis against the two-tailed alternative, then we should check if c belongs to the confidence interval:

- If $c \in \left[\widehat{\beta}_i - t_c \cdot \text{se}(\widehat{\beta}_i); \quad \widehat{\beta}_i + t_c \cdot \text{se}(\widehat{\beta}_i)\right]$, we will (most likely) **not reject** H_0 at the level of significance α.
- If $c \notin \left[\widehat{\beta}_i - t_c \cdot \text{se}(\widehat{\beta}_i); \quad \widehat{\beta}_i + t_c \cdot \text{se}(\widehat{\beta}_i)\right]$, we will (most likely) **reject** H_0.

> If the same population is sampled a number of times and interval estimates are calculated for the coefficient estimates on each sample, then $100 \cdot (1 - \alpha)\%$ of the confidence intervals would contain the true population parameters.

So, under the null hypothesis, it is quite likely that our the sample confidence interval contains the true coefficient β_i.

Continuing from where we left off...

OLS Prediction and Prediction Intervals

We have examined model specification, parameter estimation and interpretation techniques. However, usually we are not only interested in identifying and quantifying the independent variable effects on the dependent variable, but we also want to **predict** the (unknown) value of Y for any value of X.

Prediction plays an important role in financial analysis (forecasting sales, revenue, etc.), government policies (prediction of growth rates for income, inflation, tax revenue, etc.) and so on.

Let our univariate regression be defined by the linear model:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

and let assumptions (UR.1)-(UR.4) hold. Let $\widetilde{X}$ be a given value of the explanatory variable.

OLS Prediction

We want to predict the value $\widetilde{Y}$, for this given value $\widetilde{X}$. In order to do that we assume that the true DGP process remains the same for $\widetilde{Y}$.

The difference from the mean response is that when we are talking about the **prediction**, our regression outcome is composed of two parts:

$$\widetilde{\mathbf{Y}} = \mathbb{E}\left(\widetilde{\mathbf{Y}}|\widetilde{\mathbf{X}}\right) + \widetilde{\varepsilon}$$

where:

- $\mathbb{E}\left(\widetilde{Y}|\widetilde{X}\right) = \beta_0 + \beta_1\widetilde{X}$ is the **systematic component**;
- $\widetilde{\epsilon}$ - the **random component**;

The expected value of the random component is zero. We can estimate the systematic component using the OLS estimated parameters:

$$\widehat{\mathbf{Y}} = \widehat{\mathbb{E}}\left(\widetilde{\mathbf{Y}}|\widetilde{\mathbf{X}}\right) = \widetilde{\mathbf{X}}\widehat{\beta}$$

$\widehat{\mathbf{Y}}$ is called the **prediction**.

Prediction Intervals

We can defined the **forecast error** as:

$$\widetilde{\boldsymbol{e}} = \widetilde{\mathbf{Y}} - \widehat{\mathbf{Y}} = \widetilde{\mathbf{X}}\beta + \widetilde{\varepsilon} - \widetilde{\mathbf{X}}\widehat{\beta}$$

From the distribution of the dependent variable:

$$\mathbf{Y}|\mathbf{X} \sim \mathcal{N}\left(\mathbf{X}\beta,\ \sigma^2\mathbf{I}\right)$$

We know that the true observation $\widetilde{\mathbf{Y}}$ will vary with mean $\widetilde{\mathbf{X}}\beta$ and variance $\sigma^2\mathbf{I}$.

Furthermore, since $\widetilde{\varepsilon}$ are independent of $\mathbf{Y}$, it holds that (assuming that $\mathbf{X}$ and $\widetilde{\mathbf{X}}$ are fixed):

$$\begin{aligned}
\mathbb{C}\mathrm{ov}(\widetilde{\mathbf{Y}}, \widehat{\mathbf{Y}}) &= \mathbb{C}\mathrm{ov}(\widetilde{\mathbf{X}}\beta + \widetilde{\varepsilon}, \widetilde{\mathbf{X}}\widehat{\beta}) \\
&= \mathbb{C}\mathrm{ov}(\widetilde{\varepsilon}, \widetilde{\mathbf{X}}\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{Y}) \\
&= 0
\end{aligned}$$

We again highlight that $\widetilde{\varepsilon}$ are shocks in $\widetilde{\mathbf{Y}}$, which is some other realization from the DGP that is different from $\mathbf{Y}$ (which has shocks ε, and was used when estimating parameters via OLS).

Because of this, the variance of the forecast error is (assuming that $\mathbf{X}$ and $\widetilde{\mathbf{X}}$ are fixed):

$$\begin{aligned}
\mathbb{V}\mathrm{ar}\left(\widetilde{\boldsymbol{e}}\right) &= \mathbb{V}\mathrm{ar}\left(\widetilde{\mathbf{Y}} - \widehat{\mathbf{Y}}\right) \\
&= \mathbb{V}\mathrm{ar}\left(\widetilde{\mathbf{Y}}\right) - \mathbb{C}\mathrm{ov}(\widetilde{\mathbf{Y}}, \widehat{\mathbf{Y}}) - \mathbb{C}\mathrm{ov}(\widehat{\mathbf{Y}}, \widetilde{\mathbf{Y}}) + \mathbb{V}\mathrm{ar}\left(\widehat{\mathbf{Y}}\right) \\
&= \mathbb{V}\mathrm{ar}\left(\widetilde{\mathbf{Y}}\right) + \mathbb{V}\mathrm{ar}\left(\widehat{\mathbf{Y}}\right) \\
&= \sigma^2\mathbf{I} + \widetilde{\mathbf{X}}\sigma^2\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\widetilde{\mathbf{X}}^\top \\
&= \sigma^2\left(\mathbf{I} + \widetilde{\mathbf{X}}\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\widetilde{\mathbf{X}}^\top\right)
\end{aligned}$$

Note that our **prediction** interval is affected not only by the variance of the true $\widetilde{\mathbf{Y}}$ (due to random shocks), but also by the variance of $\widehat{\mathbf{Y}}$ (since coefficient estimates, $\widehat{\boldsymbol{\beta}}$, are generally imprecise and have a non-zero variance), i.e. it combines the uncertainty coming from the parameter estimates and the uncertainty coming from the randomness in a new observation.

Hence, a prediction interval will be wider than a confidence interval. In practice, we replace σ^2 with its estimator $\widehat{\sigma}^2 = \frac{1}{N-2}\sum_{i=1}^{N}\widehat{\epsilon}_i^2$.

Let $\text{se}(\widetilde{e}_i) = \sqrt{\widehat{\mathbb{V}\text{ar}}(\widetilde{e}_i)}$ be the square root of the corresponding i-th diagonal element of $\widehat{\mathbb{V}\text{ar}}(\widetilde{\boldsymbol{e}})$. This is also known as the **standard error of the forecast**. Then, the $100 \cdot (1-\alpha)\%$ **prediction interval** can be calculated as:

$$\widehat{Y}_i \pm t_{(1-\alpha/2, N-2)} \cdot \text{se}(\widetilde{e}_i)$$

Example

We will generate a univariate linear regression with $\beta_0 = 2$, $\beta_1 = 0.4$, $N = 100$ and X - an equally spaced sequence from an interval in $[0, 20]$.

```
import numpy as np
#
np.random.seed(123)
#
N = 100
beta_0 = 2
beta_1 = 0.4
#
x = np.linspace(start = 0, stop = 20, num = N)
e = np.random.normal(loc = 0, scale = 2, size = N)
y = beta_0 + beta_1 * x + e
```

Next, we will estimate the coefficients and their standard errors:

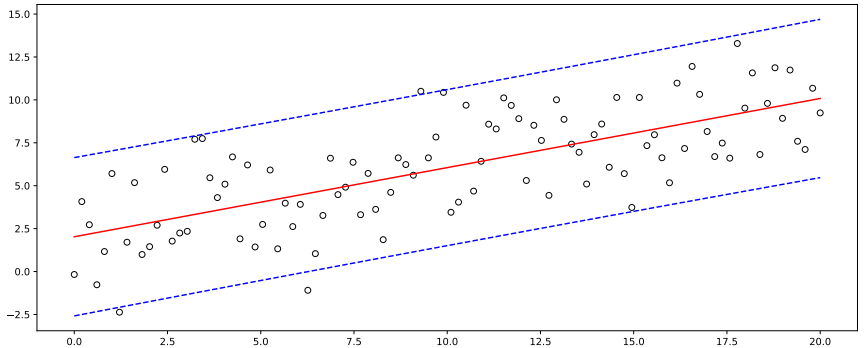
```
# Manual Calculation
#  Estimate the parameters:
x_mat = np.column_stack((np.ones(len(x)), x))
xtx = np.dot(np.transpose(x_mat), x_mat)
xty = np.dot(np.transpose(x_mat), y)
beta_mat_est = np.dot(np.linalg.inv(xtx), xty)
#  Calculate model fit:
y_fit = beta_mat_est[0] + beta_mat_est[1] * x
#  Calculate the residuals:
resid = y - y_fit
#  Estimate the standard errors:
sigma2_est = sum(resid**2) / (len(x) - 2)
var_beta = sigma2_est * np.linalg.inv(np.dot(np.transpose(x_mat), x_mat))
std_err = np.sqrt(np.diag(var_beta))
```

For simplicity, assume that we will predict Y for the existing values of X:

```
import scipy.stats as stats
import pandas as pd
#
# Let's calculate the predicted values:
x_new  = x_mat
y_pred = np.dot(x_new,  beta_mat_est)
# Calculate the prediction SE:
y_pred_se = np.linalg.inv(np.dot(np.transpose(x_mat), x_mat))
y_pred_se = np.dot(np.dot(x_new, y_pred_se), np.transpose(x_new))
y_pred_se = np.identity(len(x_new)) + y_pred_se
y_pred_se = sigma2_est * y_pred_se
y_pred_se = np.sqrt(np.diag(y_pred_se))
# Prediction intervals for the predicted Y:
y_pred_lower = y_pred - stats.t.ppf(q = 1 - 0.05 / 2, df = N-2) * y_pred_se
y_pred_upper = y_pred + stats.t.ppf(q = 1 - 0.05 / 2, df = N-2) * y_pred_se
print(pd.DataFrame(np.column_stack([y_pred, y_pred_lower, y_pred_upper])).head())
```

```
##           0         1         2
## 0  2.026150 -2.585366  6.637667
## 1  2.107526 -2.501381  6.716433
## 2  2.188901 -2.417448  6.795250
## 3  2.270276 -2.333567  6.874119
## 4  2.351651 -2.249738  6.953040
```

```
import matplotlib.pyplot as plt
#
_ = plt.figure(num = 0, figsize=(15, 6))
_ = plt.plot(x, y, linestyle = "None", marker = "o",
         markerfacecolor = "None", color = "black")
_ = plt.plot(x, y_pred, color = "red")
_ = plt.plot(x, y_pred_lower, color = "blue", linestyle = "--")
_ = plt.plot(x, y_pred_upper, color = "blue", linestyle = "--")
plt.show()
```

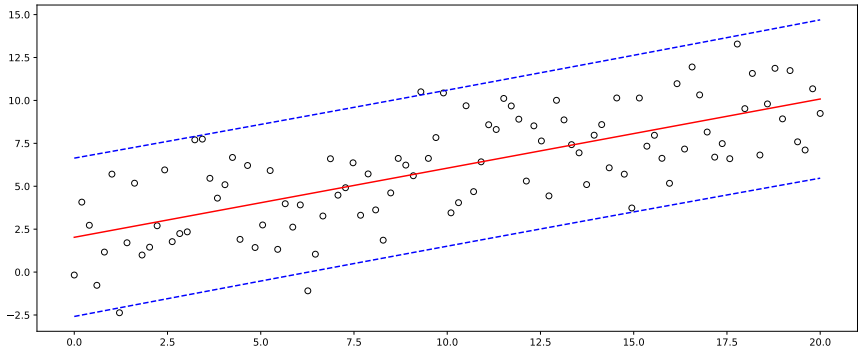


Just like for the confidence intervals, we can get the prediction intervals from the built-in functions:

```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import summary_table
#
# Automatically estimate the OLS parameters:
lm_fit = sm.OLS(y, x_mat).fit()
#
dt = lm_fit.get_prediction(x_mat).summary_frame(alpha = 0.05)
y_prd = dt['mean']
yprd_ci_lower = dt['obs_ci_lower']
yprd_ci_upper = dt['obs_ci_upper']
print(pd.DataFrame(np.column_stack([y_prd, yprd_ci_lower, yprd_ci_upper])).head())
```

```
##           0         1         2
## 0  2.026150 -2.585366  6.637667
## 1  2.107526 -2.501381  6.716433
## 2  2.188901 -2.417448  6.795250
## 3  2.270276 -2.333567  6.874119
## 4  2.351651 -2.249738  6.953040
```

```
_ = plt.figure(num = 1, figsize=(15, 6))
_ = plt.plot(x, y, linestyle = "None", marker = "o",
         markerfacecolor = "None", color = "black")
_ = plt.plot(x, y_prd, color = "red")
_ = plt.plot(x, yprd_ci_lower, color = "blue", linestyle = "--")
_ = plt.plot(x, yprd_ci_upper, color = "blue", linestyle = "--")
plt.show()
```



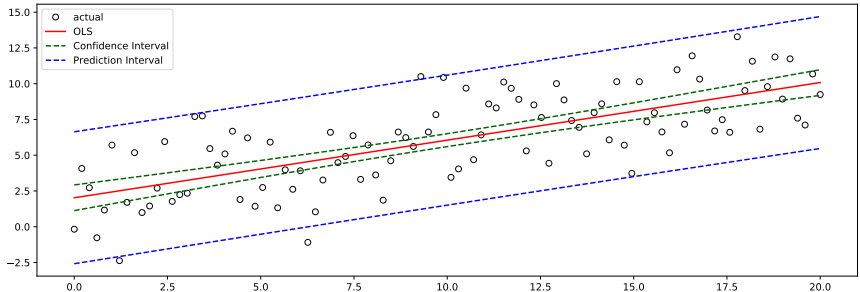
Confidence Intervals vs Prediction Intervals

Confidence intervals	**Prediction intervals**
Tell you about how well you have determined the mean.	Tell you where you can expect to see the next data point sampled.
Assume that the data really are randomly sampled from a Gaussian distribution.	Assume that the data really are randomly sampled from a Gaussian distribution.
If you sample the data many times, and calculate a confidence interval of the mean from each sample, you'd expect about 95% of those intervals to include the true value of the population mean.	Collect a sample of data and calculate a prediction interval. Then sample one more value from the population. If you do this many times, you'd expect that next value to lie within that prediction interval in 95% of the samples.
The key point is that the confidence interval tells you about the likely location of the true population parameter.	The key point is that the prediction interval tells you about the distribution of values, not the uncertainty in determining the population mean.

Prediction intervals must account for both: (i) the uncertainty of the population mean; (ii) the randomness (i.e. scatter) of the data. So, a prediction interval is always wider than a confidence interval.

```
ym_ci_lower = dt['mean_ci_lower']
ym_ci_upper = dt['mean_ci_upper']
```

```
_ = plt.figure(num = 2, figsize = (15, 5))
_ = plt.plot(x, y, linestyle = "None", marker = "o", label = "actual",
         markerfacecolor = "None", color = "black")
_ = plt.plot(x, y_fit, color = "red", label = "OLS")
_ = plt.plot(x, ym_ci_lower, color = "darkgreen", linestyle = "--", label = "Confidence Interval")
_ = plt.plot(x, ym_ci_upper, color = "darkgreen", linestyle = "--")
_ = plt.plot(x, yprd_ci_lower, color = "blue", linestyle = "--", label = "Prediction Interval")
_ = plt.plot(x, yprd_ci_upper, color = "blue", linestyle = "--")
_ = plt.legend()
plt.show()
```



Prediction intervals when Y is transformed

We will examine the following exponential model, $Y = \exp(\beta_0 + \beta_1 X + \epsilon)$, which we can rewrite as a log-linear model:

$$\log(Y) = \beta_0 + \beta_1 X + \epsilon$$

Having estimated the log-linear model we are interested in the predicted value $\widehat{Y}$. Unfortunately, our specification allows us to calculate the prediction of the log of Y, $\widehat{\log(Y)}$. Nevertheless, we can obtain the predicted values by taking the exponent of the prediction, namely:

$$\widehat{Y} = \exp\left(\widehat{\log(Y)}\right) = \exp\left(\widehat{\beta}_0 + \widehat{\beta}_1 X\right)$$

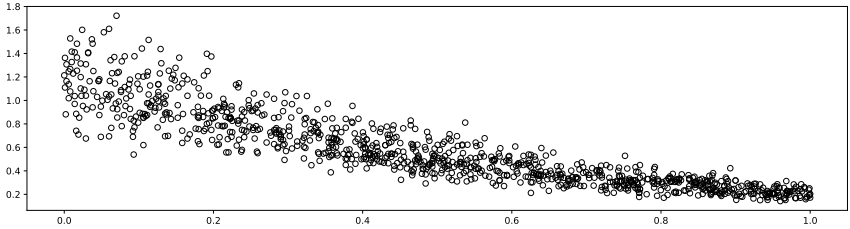
Having obtained the *point* predictor $\widehat{Y}$, we may be further interested in calculating the prediction (or, forecast) intervals of $\widehat{Y}$. In order to do so, we apply the same technique that we did for the point predictor - we estimate the prediction intervals for $\widehat{\log(Y)}$ and take their exponent.

Then, a $100 \cdot (1-\alpha)\%$ prediction interval for Y is:

$$\left[\exp\left(\widehat{\log(Y)} - t_c \cdot \text{se}(\widetilde{e}_i)\right); \quad \exp\left(\widehat{\log(Y)} + t_c \cdot \text{se}(\widetilde{e}_i)\right)\right]$$

```
np.random.seed(123)
#
N = 1000
beta_0 = 0.2
beta_1 = -1.8
#
x = np.random.choice(np.linspace(start = 0, stop = 1, num = N),
                     size = N, replace = True)
e = np.random.normal(loc = 0, scale = 0.2, size = N)
y = np.exp(beta_0 + beta_1 * x + e)
```

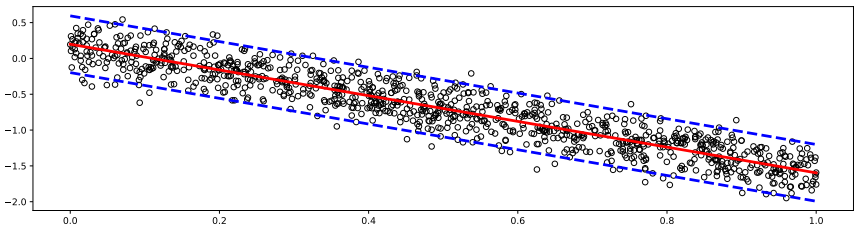
```
_ = plt.figure(num = 3, figsize = (16, 4))
_ = plt.plot(x, y, linestyle = "None", marker = "o",
        markerfacecolor = "None", color = "black")
plt.show()
```



We estimate the model via OLS and calculate the predicted values $\widehat{\log(Y)}$ and plot $\widehat{\log(Y)}$ along with their prediction intervals:

```
lm_fit = sm.OLS(np.log(y), sm.add_constant(x)).fit()
dt = lm_fit.get_prediction(sm.add_constant(x)).summary_frame(alpha = 0.05)
log_y_prd = dt['mean']
log_y_prd_lower = dt['obs_ci_lower']
log_y_prd_upper = dt['obs_ci_upper']
```

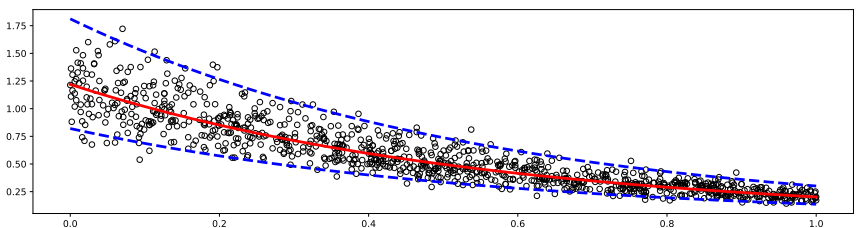
```
_ = plt.figure(num = 4, figsize=(16, 4))
_ = plt.plot(x, np.log(y), linestyle = "None", marker = "o",
         markerfacecolor = "None", color = "black")
_ = plt.plot(x[np.argsort(x)], log_y_prd[np.argsort(x)], color = "red", linewidth = 3)
_ = plt.plot(x[np.argsort(x)], log_y_prd_lower[np.argsort(x)],
         color = "blue", linestyle = "--", linewidth = 3)
_ = plt.plot(x[np.argsort(x)], log_y_prd_upper[np.argsort(x)],
         color = "blue", linestyle = "--", linewidth = 3)
plt.show()
```



Finally, we take the exponent of $\widehat{\log(Y)}$ and the prediction interval to get the predicted value and 95% prediction interval for $\widehat{Y}$:

```
y_prd = np.exp(log_y_prd)
y_prd_lower = np.exp(log_y_prd_lower)
y_prd_upper = np.exp(log_y_prd_upper)
```

```
_ = plt.figure(num = 5, figsize=(16, 4))
_ = plt.plot(x, y, linestyle = "None", marker = "o",
         markerfacecolor = "None", color = "black")
_ = plt.plot(x[np.argsort(x)], y_prd[np.argsort(x)], color = "red", linewidth = 3)
_ = plt.plot(x[np.argsort(x)], y_prd_lower[np.argsort(x)],
         color = "blue", linestyle = "--", linewidth = 3)
_ = plt.plot(x[np.argsort(x)], y_prd_upper[np.argsort(x)],
         color = "blue", linestyle = "--", linewidth = 3)
plt.show()
```



Alternatively, notice that for the log-linear (and similarly for the log-log) model:

$$Y = \exp(\beta_0 + \beta_1 X + \epsilon) = \exp(\beta_0 + \beta_1 X) \cdot \exp(\epsilon) = \mathbb{E}(Y|X) \cdot \exp(\epsilon)$$

the prediction is comprised of the **systematic** and the **random** components, but they are **multiplicative**, rather than **additive**.

Therefore we can use the properties of the log-normal distribution to derive an alternative **corrected** prediction of the log-linear model:

$$\widehat{Y}_c = \widehat{\mathbb{E}}(Y|X) \cdot \exp(\widehat{\sigma}^2/2) = \widehat{Y} \cdot \exp(\widehat{\sigma}^2/2)$$

Because, if $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$, then:

- $\mathbb{E}(\exp(\epsilon)) = \exp(\mu + \sigma^2/2)$;
- $\mathbb{V}\mathrm{ar}(\epsilon) = \left[\exp(\sigma^2) - 1\right] \exp(2\mu + \sigma^2)$.

For larger samples sizes $\widehat{Y}_c$ is closer to the true mean than $\widehat{Y}$. On the other hand, in smaller samples $\widehat{Y}$ performs better than $\widehat{Y}_c$. Finally, it also depends on the scale of X.

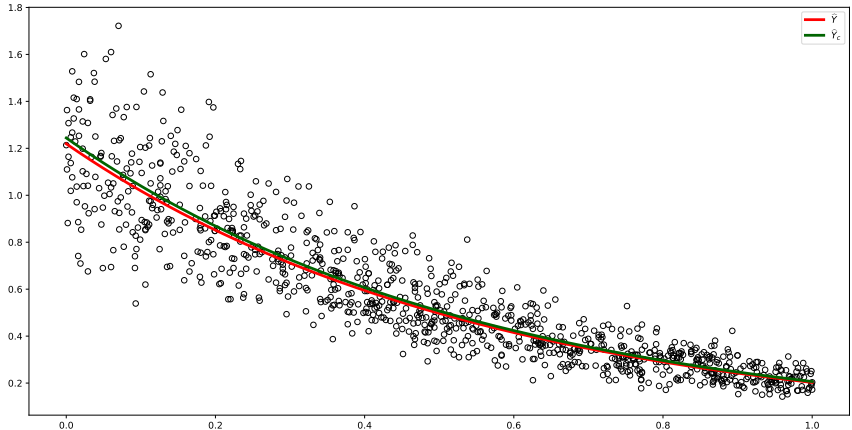
Because $\exp(0) = 1 \leq \exp(\widehat{\sigma}^2/2)$, the corrected predictor will always be larger than the natural predictor: $\widehat{Y}_c \geq \widehat{Y}$.

> Furthermore, this correction assumes that the errors have a normal distribution (i.e. that (UR.4) holds).

In our case:

```
sigma2_est = sum(lm_fit.resid**2) / (len(y) - 2)
y_prd_c  =  y_prd * np.exp(sigma2_est / 2)
```

```
_ = plt.figure(num = 6, figsize=(16, 4))
_ = plt.plot(x, y, linestyle = "None", marker = "o",
         markerfacecolor = "None", color = "black")
_ = plt.plot(x[np.argsort(x)], y_prd[np.argsort(x)],
          color = "red", linewidth = 3, label = "$\\widehat{Y}$")
_ = plt.plot(x[np.argsort(x)], y_prd_c[np.argsort(x)],
          color = "darkgreen", linewidth = 3, label = "$\\widehat{Y}_c$")
_ = plt.legend()
plt.show()
```



There is a slight difference between the *corrected* and the *natural* predictor when the variance of the sample, Y, increases.

The same ideas apply when we examine a log-log model.

Goodness-Of-Fit

Goodness-Of-Fit

Having the following model:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i, \quad i = 1, ..., N$$

Allows us to:

1. Explain, how the dependent variable Y_i changes, if the independent variable X_i changes.
2. Predict the value of $\widetilde{Y}$, given a value of $\widetilde{X}$.

In order to have an accurate prediction of Y, we hope that the independent variable X helps us *explain* as much variation in Y as possible (hence why X is usually referred to as an *explanatory* variable). Ideally, the variance of X will help *explain* the variance in Y.

Having said that, we would like to have a way to **measure** just how *good* our model is - how much of the variation in Y can be explained by the variation in X using our model - we need a **goodness-of-fit** measure.

Another way to look at it is - a **goodness-of-fit** measure aims to quantify how well the estimated model *fits* the data. Fortunately, there are many ways to measure the goodness-of-fit of the estimated model.

Model Residuals: RSS, ESS and TSS

We can separate our univariate regression into two components:

$$Y_i = \mathbb{E}(Y_i|X_i) + \epsilon_i$$

where:

- $\mathbb{E}(Y_i|X_i) = \beta_0 + \beta_1 X_i$ is the explainable, *systematic*, component of our model;
- ϵ_i is the random, **unsystematic**, unexplainable component of our model.

In practical application, we do not observe the true systematic and the true random components, but we can use the OLS to estimate the unknown parameters. then our regression can be written as:

$$Y_i = Y_i \pm \widehat{Y}_i = \widehat{Y}_i + \widehat{\epsilon}_i$$

where $\widehat{Y}_i = \widehat{\beta}_0 + \widehat{\beta}_1 X_i$ and $\widehat{\epsilon}_i = Y_i - \widehat{Y}_i$.

Because the least squares fitted regression passed through the sample mean $(\overline{Y}, \overline{X})$, if we subtract the sample mean of Y, $\overline{Y} = \frac{1}{N}\sum_{i=1}^{N} Y_i$ from both sides of the equation, we rewrite our model in terms of **differences (i.e. variation) from the process mean**:

$$Y_i - \overline{Y} = (\widehat{Y}_i - \overline{Y}) + \widehat{\epsilon}_i$$

This expression states that the difference between Y_i and its sample mean, $\overline{Y}$, consists of an explained, $(\widehat{Y}_i - \overline{Y})$, and unexplained, $\widehat{e}_i$, part. Taking the squares of both sides and summing across $i = 1, ..., N$ yields:

$$\sum_{i=1}^{N} (Y_i - \overline{Y})^2 = \sum_{i=1}^{N} \left((\widehat{Y}_i - \overline{Y}) + \widehat{\epsilon}_i\right)^2 = \sum_{i=1}^{N} \left(\widehat{Y}_i - \overline{Y}\right)^2 + 2\sum_{i=1}^{N} \left(\widehat{Y}_i - \overline{Y}\right)\widehat{\epsilon}_i + \sum_{i=1}^{N} \widehat{\epsilon}_i^2$$

Using the fact that :

$$\sum_{i=1}^{N} \left(\widehat{Y}_i - \overline{Y}\right)\widehat{\epsilon}_i = \sum_{i=1}^{N} \left(\widehat{\beta}_0 + \widehat{\beta}_1 X_i\right)\widehat{\epsilon}_i - \overline{Y}\sum_{i=1}^{N} \widehat{\epsilon}_i = \widehat{\beta}_0 \sum_{i=1}^{N} \widehat{\epsilon}_i + \widehat{\beta}_1 \sum_{i=1}^{N} X_i\widehat{\epsilon}_i - \overline{Y}\sum_{i=1}^{N} \widehat{\epsilon}_i = 0$$

we can rewrite the equality as:

$$\sum_{i=1}^{N} (Y_i - \overline{Y})^2 = \sum_{i=1}^{N} \left(\widehat{Y}_i - \overline{Y}\right)^2 + \sum_{i=1}^{N} \widehat{\epsilon}_i^2 \qquad (1)$$

The (1) equation gives us a decomposition of the total sample variation, into explained and unexplained components.

Define the following:

- **Total Sum of Squares (SST or TSS)** as:

$$\text{TSS} = \sum_{i=1}^{N} (Y_i - \overline{Y})^2$$

  It is a measure of *total variation* in Y around the sample mean.
- **Explained Sum of Squares (ESS)** as:

$$\text{ESS} = \sum_{i=1}^{N} (\widehat{Y}_i - \overline{Y})^2$$

  It is the part of the total variation in Y around the sample mean, that **is explained by our regression**. This is sometimes called the model sum of squares or sum of squares due to regression (which is confusingly also abbreviated as **"SSR"**).
- **Residual Sum of Squares (SSR or RSS)** as:

$$\text{RSS} = \sum_{i=1}^{N} \widehat{\epsilon}_i^2$$

  It is the part of the total variation in Y around the sample mean that is **not explained by our regression**. This is sometimes called the unexplained sum of squares or the sum of squared estimate of errors (**SSE**).

Then, (1) equation can be written simply as:

$$\text{TSS} = \text{ESS} + \text{RSS}$$

R-squared, R^2

It is often useful to compute a number that summarizes how well the OLS regression fits the data. This measure is called the **coefficient of determination**, R^2, which is the ratio of explained variation, compared to the total variation, i.e. the proportion of variation in Y that is explained by X in our regression model:

$$R^2 = \frac{\text{ESS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

- The closer R^2 is to 1, the closer the sample values of Y_i are to the fitted values $\widehat{Y}$ of our regression. Ir $R^2 = 1$, then all the sample data fall exactly on the fitted regression. In such a case our model would be a *perfect fit* for our data.
- If the sample data of Y and X do not have a linear relationship, then $R^2 = 0$ of a univariate regression.
- Values $0 < R^2 < 1$, the interpretation of R^2 is as *the proportion of the variation in Y around its mean, that is explained by the regression model*. For example $R^2 = 0.17$ means that 17% of the variation in Y is explained by X.

When comparing RSS of different models, we want to choose the model, which better fits our data.

If we want to choose a model based on its R^2 value we should note a couple of things:

- R^2 comparison **is not valid** for comparing models, that **do not have have the same transformation of the dependent variable**, for example two models - one with Y and the other with $\log(Y)$ dependent variables cannot be compared via R^2.
- R^2 does not measure the predictability power of the model. For example, a linear model may be a good fit for the data, but its forecasts may not make economic sense (e.g. forecasting negative wage for low values of years in education via a simple linear model).
- R^2 is based on the sample data, so it says nothing whether our model is close to the true population DGP.
- R^2 may be low if: the error variance, σ^2, is large; or if the variance of X is small.
- R^2 may be large even if the model is wrong. For example, even if the true relationship is non-linear, a linear model may have a larger R^2, compared to the quadratic, or even the log-linear model.

On the other hand, **the goodness-of-fit** of the model does not depend on the unit of measurement of our variables (e.g. dollars vs thousands of dollars). Furthermore, comparisons of R^2 are valid, if we compare a simple linear model to a linear-log model, as they both have the same dependent variable, Y.

In any case, a model should not be chosen **only** on the basis of model fit with R^2 as the criterion.

```
import numpy as np
#
np.random.seed(123)
#
N = 100
beta_0 = 2
beta_1 = 0.4
#
x = np.linspace(start = 0, stop = 20, num = N)
e = np.random.normal(loc = 0, scale = 2, size = N)
y = beta_0 + beta_1 * x + e
```

Next, we will estimate the coefficients.

We will use to built-in functions as we have already, plentifully, shown how the coefficients, standard errors, fitted values and residuals can be calculated manually:

```
import statsmodels.api as sm
#
lm_fit = sm.OLS(y, sm.add_constant(x)).fit()
print(lm_fit.params)
```

```
## [2.02615049 0.40280677]
```

Next, we will use the residuals to calculate the TSS, RSS and R^2:

```
RSS = np.sum(lm_fit.resid**2)
TSS = np.sum((y - np.mean(y))**2)
R_sq = 1 - RSS / TSS
print(R_sq)
```

```
## 0.5200895667266314
```

Which we can also conveniently extract from the estimated model objects:

```
print(lm_fit.rsquared)
```

```
## 0.5200895667266314
```

Finally, we may look at the full summary output of our models:

```
print(lm_fit.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:                      y   R-squared:                       0.520
## Model:                            OLS   Adj. R-squared:                  0.515
## Method:                 Least Squares   F-statistic:                     106.2
## Date:                Wed, 02 Oct 2019   Prob (F-statistic):           2.63e-17
## Time:                        20:47:44   Log-Likelihood:                -223.27
## No. Observations:                 100   AIC:                             450.5
## Df Residuals:                      98   BIC:                             455.8
## Df Model:                           1
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## const          2.0262      0.452      4.478      0.000       1.128       2.924
## x1             0.4028      0.039     10.306      0.000       0.325       0.480
## ==============================================================================
## Omnibus:                        2.753   Durbin-Watson:                   1.975
## Prob(Omnibus):                  0.252   Jarque-Bera (JB):                1.746
## Skew:                           0.035   Prob(JB):                        0.418
## Kurtosis:                       2.356   Cond. No.                         23.1
## ==============================================================================
##
## Warnings:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

and see a variety of familiar statistics.

R^2 and variable scaling

If we decide to scale the variables, by say, dividing by 10, then the R^2 would be unchanged:

```
lm_fit_scale_y = sm.OLS(y/10, sm.add_constant(x)).fit()
print(lm_fit_scale_y.params)
```

```
## [0.20261505 0.04028068]
```

```
print(lm_fit_scale_y.rsquared)
```

```
## 0.5200895667266313
```

```
lm_fit_scale_x = sm.OLS(y, sm.add_constant(x/10)).fit()
print(lm_fit_scale_x.params)
```

```
## [2.02615049 4.02806766]
```

```
print(lm_fit_scale_x.rsquared)
```

```
## 0.5200895667266314
```

```
lm_fit_scale_yx = sm.OLS(y/10, sm.add_constant(x/10)).fit()
print(lm_fit_scale_yx.params)
```

```
## [0.20261505 0.40280677]
```

```
print(lm_fit_scale_yx.rsquared)
```

```
## 0.5200895667266314
```

Cases When R^2 is Negative

A Case of a negative R^2 can arise when:

1. The predictions that are being compared to the corresponding outcomes have not been derived from a model-fitting procedure using those data. E.g. if we try to *guess* the coefficient values - e.g. we assume that coefficients of models on similar data, or in similar countries would be the same for our data;
2. We do not include an intercept, β_0 in our linear regression model;
3. When a non-linear function is used to fit the data;

In cases where negative R^2 values arise, the **mean of the data** provides a better fit to the outcomes, rather than the fitted model values, **according to this criterion,** R^2. We will later see, that there is a variety of different alternative criterions for evaluating the accuracy of a model.

We will look at each case separately.

Fitted values are not derived from the data, which is being analysed

Let's say that we use a model, which was fitted on **the following dataset**:

```
np.random.seed(123)
#
N = 1000
#
x0 = np.random.choice(np.linspace(start = 0, stop = 2, num = N), size = N, replace = True)
e0 = np.random.normal(loc = 0, scale = 1, size = N)
y0 = -2 + 2 * x0 + e0
```

The estimated model of such a dataset is:

```
lm_fit0 = sm.OLS(y0, sm.add_constant(x0)).fit()
print(lm_fit0.params)
```

```
## [-2.00889979  2.01142461]
```

Now, assume that we are analyzing a **different data sample**. Let's say that **our** data sample comes from the following underlying DGP:

```
np.random.seed(456)
#
N = 1000
beta_0 = 2
beta_1 = -2
#
x_other = np.random.choice(np.linspace(start = 0, stop = 2, num = N), size = N, replace = True)
e_other = np.random.normal(loc = 0, scale = 1, size = N)
y_other = beta_0 + beta_1 * x_other + e_other
```

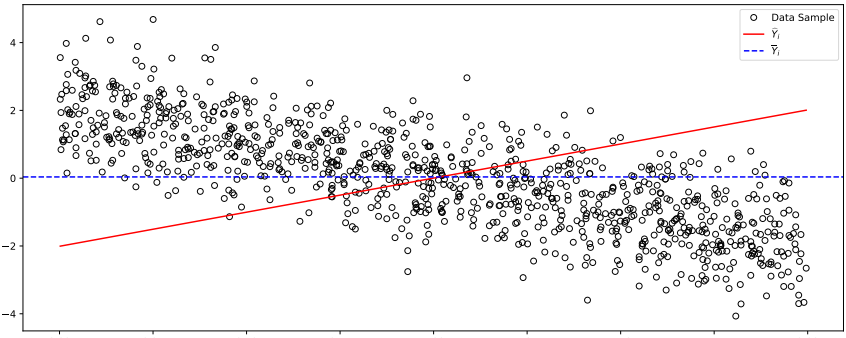
However, we make the **incorrect assumption that our data sample comes from the same population as the previous data**. This leads us to calculating the fitted values, residuals and R^2 using pre-estimated coefficients:

```
y_fit_other = lm_fit0.params[0] + lm_fit0.params[1] * x_other
resid_other = y_other - y_fit_other
#
RSS = np.sum(np.array(resid_other)**2)
TSS = np.sum((y_other - np.mean(y_other))**2)
R_sq = 1 - RSS / TSS
print(R_sq)
```

```
## -1.7715352852594477
```

Visual inspection reveals that our assumption that an existing model of one dataset is *good enough* for our dataset was **incorrect** - it is clear that our dataset is from a different DGP. For comparison, we also plot the process mean.

```
_ = plt.figure(num = 1, figsize = (16, 4))
_ = plt.plot(x_other, y_other, linestyle = "None", marker = "o", markerfacecolor = "None",
         color = "black", label = "Data Sample")
_ = plt.plot(x_other[np.argsort(x_other)], y_fit_other[np.argsort(x_other)], linestyle = "-",
         color = "red", label = "$\\widehat{Y}_i$")
_ = plt.axhline(y = np.mean(y_other), linestyle = "--", color = "blue",
         label = "$\\overline{Y}_i$")
_ = plt.legend()
plt.show()
```



If we compare models from datasets of different countries, different firms, we would run into such problems.

For example, if one firm is very large, while another is relatively new and small - making an assumption that a model on the data of one firm can be applied to the data of this new firm would be incorrect - some variables may have similar effects, but they would most likely not be the same in magnitude.

Regression without an intercept

We will generate an OLS model with an intercept

```
np.random.seed(123)
#
N = 100
beta_0 = 30
beta_1 = 2
#
x = np.linspace(start = 0, stop = 20, num = N)
e = np.random.normal(loc = 0, scale = 1, size = N)
y = beta_0 + beta_1 * x + e
```

But we will estimate the parameters of a regression model **without** an intercept. The estimated coefficient, fitted values and residuals are calculated as follows (**take note that we do not include a constant in the independent variable matrix**):

```
lm_fit = sm.OLS(y, x).fit()
print(lm_fit.params)
```

```
## [4.24107257]
```

Which results in the following negative R^2:

```
RSS = np.sum(np.array(lm_fit.resid)**2)
TSS = np.sum((y - np.mean(y))**2)
R_sq = 1 - RSS / TSS
print(R_sq)
```

```
## -0.6718501471478293
```

For cases when a model does not have an intercept, R^2 is usually computed as:

$$R^2 = 1 - \frac{RSS}{\sum_{i=1}^{N} Y_i^2}$$

where the denominator acts as if we assume that $\mathbb{E}(Y) = 0$ (and hence we assume that $\overline{Y} \approx 0$).

Applying this expression for the R^2 yields:

```
R_sq = 1 - RSS/np.sum(np.array(y)**2)
print(R_sq)
```

```
## 0.9129369975970213
```

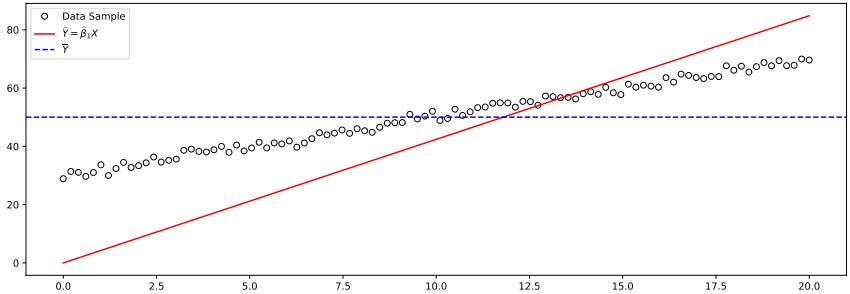
Furthermore, this value of R^2 is (**silently**) applied in the built-in OLS estimation functions:

```
print(lm_fit.rsquared)
```

```
## 0.9129369975970213
```

Unfortunately, if R^2 is calculated in this way, it ignores a very important fact about our model - a negative R^2 indicates that the regression is actually **worse** than a simple average of the process. In fact, the modified R^2 shows a very high value - the complete opposite of what we would expect to see in such a situation.

Visually, we can see that our model provides quite a poor fit. For comparison, we also plot the process mean:



So, while the *modified* R^2 *seems* high, in reality the model provides a poor fit for the data sample.

A Nonlinear function is used to fit the data with large error variance

As an example we will simulate data from the following log-linear model: $\log(Y) = \beta_0 + \beta_1 X + \epsilon$, where $\beta_0 = 0.2$, $\beta_1 = 2$, $N = 100$, $\epsilon \sim \mathcal{N}(0, 1)$, and X is a random sample with replacement from an interval from 0 to 0.5, equally spaced into N elements.

```
np.random.seed(123)
#
N = 100
beta_0 = 0.2
beta_1 = 2
#
x = np.random.choice(np.linspace(start = 0, stop = 0.5, num = N), size = N, replace = True)
e = np.random.normal(loc = 0, scale = 1, size = N)
y = np.exp(beta_0 + beta_1 * x + e)
```

This data has a small variation in X and a (relative to the variance in $\log(Y)$ and in ϵ) large error variance:

```
print(np.var(x))
```

```
## 0.023800418324660753
```

```
print(np.var(np.log(y)))
```

```
## 1.1624704319571753
```

```
print(np.var(e))
```

```
## 1.0245932457731062
```

If we estimate the **correct** model and look at the coefficients and R^2:

```
lm_fit = sm.OLS(np.log(y), sm.add_constant(x)).fit()
print(lm_fit.params)
```

```
## [0.0878827  2.44826432]
```

```
print(lm_fit.rsquared)
```

```
## 0.12272111156714938
```

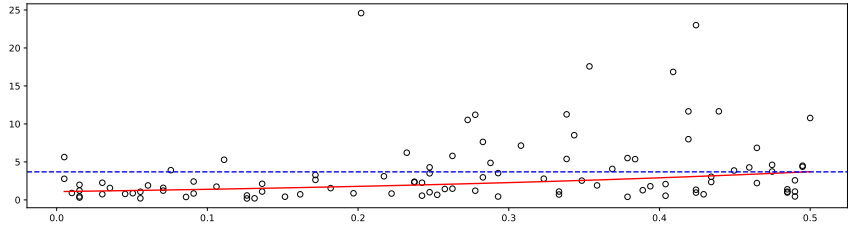
We see that the R^2 is **very small**. Furthermore, if we were to back-transform our fitted values and calculate R^2:

```
y_fit = np.exp(lm_fit.fittedvalues)
resid = y - y_fit
#
RSS = np.sum(resid**2)
TSS = np.sum((y - np.mean(y))**2)
R_sq = 1 - RSS/TSS
print(R_sq)
```

```
## -0.04781349315972472
```

We see that it is even **worse**.

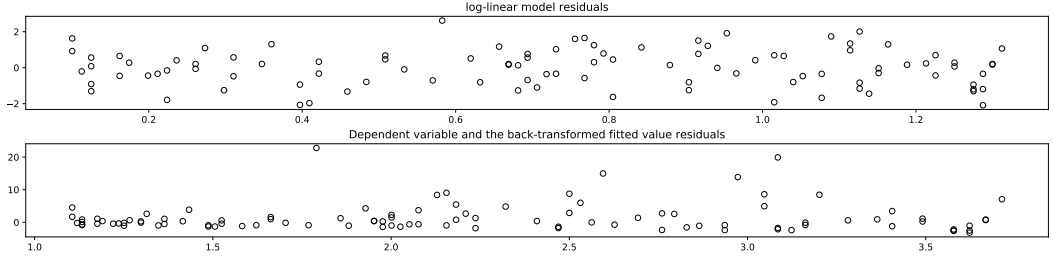
The plot of the fitted values:



So, a large variance of the error term, or a small variance of the independent variable, will result in a lower R^2 value overall. Furthermore, back-transforming would likely result in a *lower* R^2 value.

Finally, plotting the residuals against the fitted values indicates that the residuals of $Y - \exp(\widehat{\log(Y)})$ do not have the same properties as the log-linear model residuals:

```
fig = plt.figure(num = 4, figsize = (16, 8))
_ = fig.add_subplot('211').plot(lm_fit.fittedvalues, lm_fit.resid, color = "black",
        linestyle = "None", marker = "o", markerfacecolor = 'None')
_ = plt.title("log-linear model residuals")
_ = fig.add_subplot('212').plot(y_fit, resid, color = "black",
        linestyle = "None", marker = "o", markerfacecolor = 'None')
_ = plt.title("Dependent variable and the back-transformed fitted value residuals")
_ = plt.tight_layout()
plt.show()
```



Correlation Analysis

The correlation coefficient between X and Y is defined as:

$$\rho_{X,Y} = \frac{\mathbb{C}\mathrm{ov}(X,Y)}{\sqrt{\mathbb{V}\mathrm{ar}(X)}\sqrt{\mathbb{V}\mathrm{ar}(Y)}} = \frac{\sigma_{X,Y}}{\sigma_X \sigma_Y}$$

The *sample* correlation is calculated as:

$$r_{X,Y} = \frac{\widehat{\sigma}_{X,Y}}{\widehat{\sigma}_X \widehat{\sigma}_Y} = \frac{\frac{1}{N-1}\sum_{i=1}^{N}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(X_i - \overline{X})^2}\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(Y_i - \overline{Y})^2}}$$

The sample correlation $-1 \leq r_{X,Y} \leq 1$ measures the strength of the **linear association** between the sample values of X and Y.

Correlation Analysis and R^2

There is a relationship between R^2 and $r_{X,Y}$:

1. $R^2 = r^2_{X,Y}$. So, R^2 can be computed as the square of the sample correlation between Y and X.
2. $R^2 = r^2_{Y,\widehat{Y}}$. So, R^2 can be computed as the square of the sample correlation between Y and its fitted values $\widehat{Y}$. As such, R^2 measures the linear association, the *goodness-of-fit*, between the sample data Y, and its predicted values $\widehat{Y}$. Because of this, R^2 is sometimes called a measure of *goodness-of-fit*.

```
print(lm_fit.rsquared)
```

```
## 0.12272111156714938
```

```
print(np.corrcoef(np.log(y), x)[0][1]**2)
```

```
## 0.12272111156714918
```

```
print(np.corrcoef(np.log(y), lm_fit.fittedvalues)[0][1]**2)
```

```
## 0.12272111156714925
```

A General (pseudo) R-squared Measure, R_g^2

As we have seen, we may need to *back*-transform our independent variable. Then, we can calculate a general (pseudo) measure of R^2:

$$R_g^2 = r^2_{Y,\widehat{Y}} = \mathbb{C}\text{orr}(Y, \widehat{Y})^2$$

```
print(np.corrcoef(np.log(y), lm_fit.fittedvalues)[0][1]**2)
```

```
## 0.12272111156714925
```

```
print(np.corrcoef(y, y_fit)[0][1]**2)
```

```
## 0.05975262825731168
```

In our previous example we can calculate R_g^2 for both the log and the back-transformed values:

A way to look at it is that R^2 measures the **variation explained by our model**, whereas R_g^2 measures the **variance explained by our model**. In a linear regression, the two definitions are the same, as long as the intercept coefficient is included in the model.

Regression Diagnostics

Regression Diagnostics

In many cases while carrying out statistical/econometric analysis, we are not sure, whether we have correctly specified our model. As we have seen, the R^2 can be artificially small (or large), regardless of the specified model. As such, there are a number of regression diagnostics and specification tests.

For the univariate regression, the most crucial assumptions come from **(UR.3)** and **(UR.4)**, namely:

- $\mathbb{V}\text{ar}(\epsilon_i|\mathbf{X}) = \sigma_\epsilon^2, \ \forall i = 1, .., N$
- $\mathbb{C}\text{ov}(\epsilon_i, \epsilon_j) = 0, \ i \neq j$
- $\varepsilon|\mathbf{X} \sim \mathcal{N}\left(\mathbf{0}, \sigma_\epsilon^2 \mathbf{I}\right)$

We note that the residuals are defined as:

$$\begin{aligned}
\widehat{\varepsilon} &= \mathbf{Y} - \widehat{\mathbf{Y}} \\
&= \mathbf{Y} - \mathbf{X}\widehat{\beta} \\
&= \mathbf{Y} - \mathbf{X}\left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{Y} \\
&= \left[\mathbf{I} - \mathbf{X}\left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top\right] \mathbf{Y}
\end{aligned}$$

Hence, for the OLS **residuals** (i.e. not the true unobserved errors) the expected value of the residuals is still zero:

$$\begin{aligned}
\mathbb{E}\left(\widehat{\boldsymbol{\varepsilon}}|\mathbf{X}\right) &= \mathbb{E}\left(\left[\mathbf{I}-\mathbf{X}\left(\mathbf{X}^{\top}\mathbf{X}\right)^{-1}\mathbf{X}^{\top}\right]\mathbf{Y}|\mathbf{X}\right) \\
&= \mathbb{E}\left(\left[\mathbf{I}-\mathbf{X}\left(\mathbf{X}^{\top}\mathbf{X}\right)^{-1}\mathbf{X}^{\top}\right]\left(\mathbf{X}\boldsymbol{\beta}+\boldsymbol{\varepsilon}\right)|\mathbf{X}\right) \\
&= \mathbf{X}\boldsymbol{\beta}+\mathbb{E}(\boldsymbol{\varepsilon})-\mathbf{X}\boldsymbol{\beta}-\mathbf{X}\left(\mathbf{X}^{\top}\mathbf{X}\right)^{-1}\mathbf{X}^{\top}\mathbb{E}(\boldsymbol{\varepsilon}) \\
&= 0
\end{aligned}$$

or simplicity, let $\widehat{\boldsymbol{\varepsilon}} = [\mathbf{I}-\mathbf{H}]\,\mathbf{Y}$, where $\mathbf{H} = \mathbf{X}\left(\mathbf{X}^{\top}\mathbf{X}\right)^{-1}\mathbf{X}^{\top}$.

Consequently, the variance-covariance matrix of the residuals is:

$$\begin{aligned}
\mathbb{V}\mathrm{ar}\left(\widehat{\boldsymbol{\varepsilon}}|\mathbf{X}\right) &= \mathbb{V}\mathrm{ar}\left([\mathbf{I}-\mathbf{H}]\,\mathbf{Y}|\mathbf{X}\right) \\
&= [\mathbf{I}-\mathbf{H}]\,\mathbb{V}\mathrm{ar}\left(\mathbf{Y}|\mathbf{X}\right)[\mathbf{I}-\mathbf{H}]^{\top} \\
&= [\mathbf{I}-\mathbf{H}]\,\sigma^{2}\,[\mathbf{I}-\mathbf{H}]^{\top} \\
&= \sigma^{2}\left[\mathbf{I}-\mathbf{H}^{\top}-\mathbf{H}+\mathbf{H}\mathbf{H}^{\top}\right] \\
&= \sigma^{2}\left[\mathbf{I}-\mathbf{H}^{\top}-\mathbf{H}+\mathbf{H}^{\top}\right] \\
&= \sigma^{2}\left[\mathbf{I}-\mathbf{H}\right]
\end{aligned} \tag{2}$$

This result shows an important distinction of the residuals from the errors - the residuals may have different variances (which are the diagonal elements of $\mathbb{V}\mathrm{ar}\left(\widehat{\varepsilon}|\mathbf{X}\right)$), even if the true errors (which affect the process $\mathbf{Y}$) all have the same variance σ^2.

Residual Diagnostic Plots

One way to examine the adequacy of the model is to visualize the residuals. There are a number of ways to do this:

- Plotting the residuals $\widehat{\epsilon}_i$ against the fitted values $\widehat{Y}_i$;
- Plotting the residuals $\widehat{\epsilon}_i$ against X_i
- Plotting the residual Q-Q plot, histogram or boxplot.

In all cases, if there are no violations of our (UR.2) or (UR.3) assumptions - the plots should reveal **no patterns**. The residual histogram, Q-Q plot should be approximately normal so that our assumption (UR.4) holds.

As we are not guaranteed to specify a correct functional form, residual plots offer a great insight on what possible functional form we may have missed.

We should note that when having multiple models, it is only meaningful to compare the residuals of models with the same dependent variable. For example, comparing the residuals of a linear-linear model (with Y) and of a log-linear model (with $\log(Y)$) is **not** meaningful as they have different value scales.

Transforming the dependent or the independent variables may help to alleviate some of the problems of the residuals:

- If nonlinearities are present in the residual plots - we must firstly account for them, and only after can we check, whether the errors have a constant variance.
- Transforming **Y** primarily aims to help with problems with the error terms (and may help with non-linearity);
- Transforming **X** primarily aims to help with correcting for non-linearity;
- Sometimes transforming **X** is enough to account for non-linearity and have normally distributed errors, while transforming **Y** may account for non-linearity but might make the errors non-normally distributed.
- Other times, transforming **X** does not help account for the nonlinear relationship at all;

Remember that the Q-Q plot plots quantiles of the data versus quantiles of a distribution. If the observations come from a normal distribution we would expect the observed order statistics plotted against the expected (theoretical) order statistics to form an approximately straight line.

Example

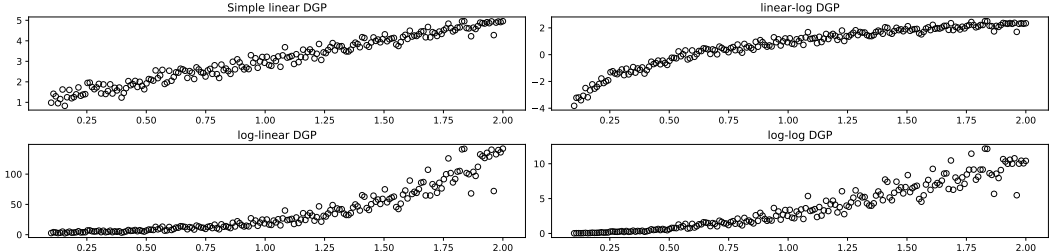
We will generate four different models:

- a simple linear model: $Y = \beta_0 + \beta_1 X + \epsilon$;
- a log-linear model: $\log(Y) = \beta_0 + \beta_1 X + \epsilon$;
- a linear-log model: $Y = \beta_0 + \beta_1 \log(X) + \epsilon$;
- a log-log model: $\log(Y) = \beta_0 + \beta_1 \log(X) + \epsilon$;

For each case, we will estimate a simple linear model on the data and examine the residual plots. For simplicity, we will use the same X_i and the same $\epsilon_i \sim \mathcal{N}(0, 0.2^2)$, $i = 1, ..., N$ with $N = 200$, $\beta_0 = 1$ and $\beta_1 = 2$.

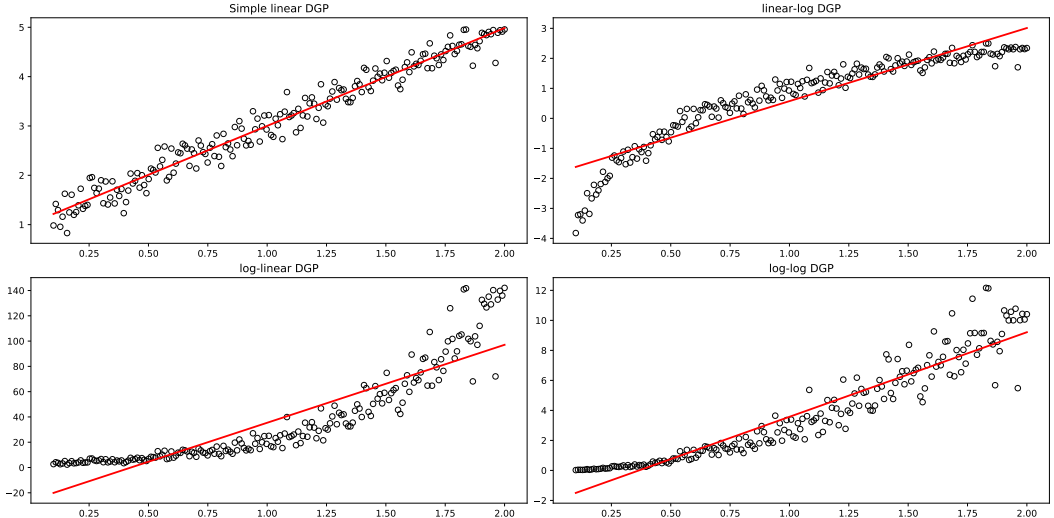
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import scipy.stats as stats
#
np.random.seed(123)
# Sample size and coefficients
N = 200
beta_0 = 1
beta_1 = 2
# Variables which will be the same for each model:
x = np.linspace(start = 0.1, stop = 2, num = N)
e = np.random.normal(loc = 0, scale = 0.2, size = N)
```

```python
# Simple linear model
y = beta_0 + beta_1 * x + e
data_lin = pd.DataFrame([y, x, e], index = ["y", "x", "e"]).T
# Linear-Log model:
y = beta_0 + beta_1 * np.log(x) + e
data_linlog = pd.DataFrame([y, x, e], index = ["y", "x", "e"]).T
# Log-linear model:
y = np.exp(beta_0 + beta_1 * x + e)
data_loglin = pd.DataFrame([y, x, e], index = ["y", "x", "e"]).T
# Log-Log model:
y = np.exp(beta_0 + beta_1 * np.log(x) + e)
data_loglog = pd.DataFrame([y, x, e], index = ["y", "x", "e"]).T
```

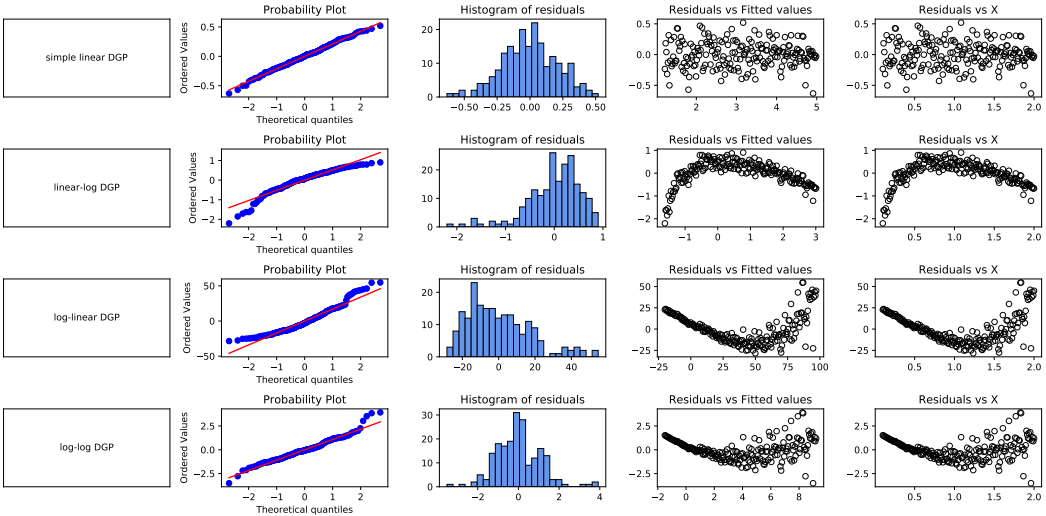


Next, we will estimate the simple linear regression for each dataset:

```
mdl1 = sm.OLS(data_lin["y"], sm.add_constant(data_lin["x"])).fit()
mdl2 = sm.OLS(data_linlog["y"], sm.add_constant(data_linlog["x"])).fit()
mdl3 = sm.OLS(data_loglin["y"], sm.add_constant(data_loglin["x"])).fit()
mdl4 = sm.OLS(data_loglog["y"], sm.add_constant(data_loglog["x"])).fit()
```



Then, the different residual plots are as follows:



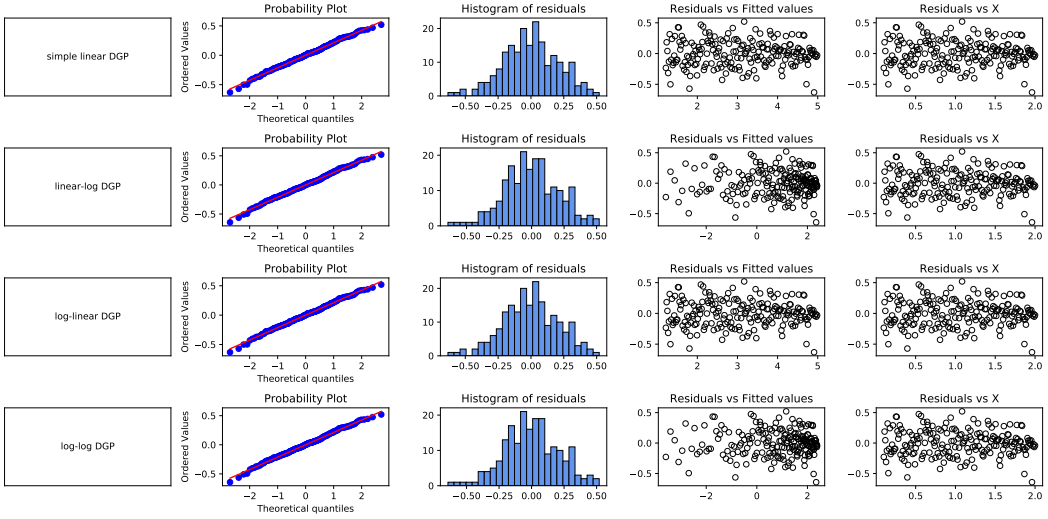
We see that the linear model for the dataset, which is generated from a simple linear DGP, has residuals which appear to be random - we do not see any non-random patterns in the scatterplots.

Furthermore, the histogram and Q-Q plot indicate that the residuals may be from a normal distribution.

On the other hand, a simple linear model does not fit the data well, if the data is sampled from a non-linear DGP - we see clear patterns in the residual scatter plots, as well as non-normality.

For comparison, if we were to fit the **correct** models, we would have the following plots:

```
mdl2_correct = sm.OLS(data_linlog["y"], sm.add_constant(np.log(data_linlog["x"]))).fit()
mdl3_correct = sm.OLS(np.log(data_loglin["y"]), sm.add_constant(data_loglin["x"])).fit()
mdl4_correct = sm.OLS(np.log(data_loglog["y"]), sm.add_constant(np.log(data_loglog["x"]))).fit()
```



Then the residuals are normally distributed and do not have any patterns or change in variance.

Residual Heteroskedasticity

If $\mathbb{V}\mathrm{ar}(\epsilon_i|\mathbf{X}) = \sigma^2_\epsilon,\ \forall i = 1, .., N$, we say that the residuals are **homoskedastic**. If this assumption is violated, we say that the residuals are **heteroskedastic** - that is, their variance is not constant throughout observations.

The consequences of heteroskedasticity are as follows:

- OLS parameters remain **unbiased**;
- OLS estimates are no longer efficient (i.e. they no longer have the smallest variance). The reason for this is that OLS gives equal weight to all observations in the data, when in fact, observation with larger error variance contain less information, compared to observations with smaller error variance;
- The variance estimate of the residuals **is biased**, and hence the standard errors are **biased**. This in turn leads to a bias in test statistics and confidence intervals.
- Because of standard error bias, we may fail to reject the null hypothesis whether $\beta_i = 0$ in our estimated model, when the null hypothesis is actually false (i.e. making a Type II error).

There are a few possible corrections to account for heteroskedasticity:

- Take logarithms of the data, this may be able to help **linearize** the data and in turn, the residuals;
- Apply a different estimation method. We will examine this later on, but one possibility is to use a **Weighted Least Squares** estimation method, which gives different observations different weights and allows to account for a non-constant variance;
- It is possible to correct the the biased standard errors for heteroskedasticity. This would leave the OLS estimates unchanged. White's heteroskedasticity-consistent standard errors (or, robust standard errors) give a consistent variance estimator.

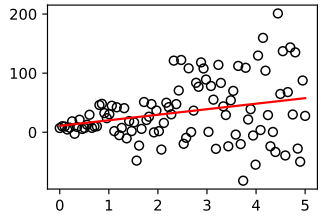
Example

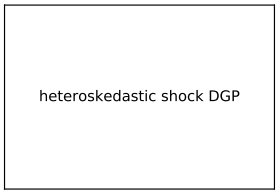
We are going to simulate the following model:

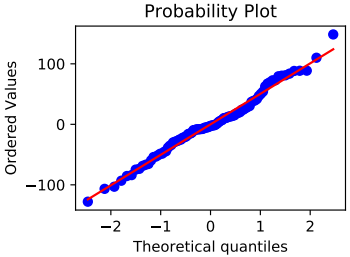
$$Y_i = \beta_0 + \beta_1 X_i + u_i$$
$$u_i = \sqrt{i} \cdot \epsilon_i, \text{ where } \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

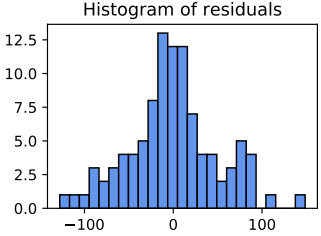
```
np.random.seed(123)
#
N = 100
beta_0 = 8
beta_1 = 10
#
x = np.linspace(start = 0, stop = 5, num = N)
e = np.random.normal(loc = 0, scale = 0.8, size = N)
u = np.array(list(range(1, N + 1))) * e
#
y = beta_0 + beta_1 * x + u
#
mdl = sm.OLS(y, sm.add_constant(x)).fit()
print(mdl.summary().tables[1])
```

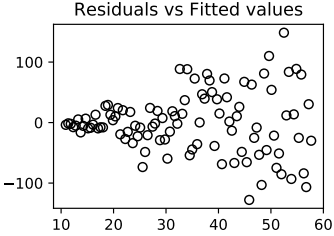
```
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## const         10.8945      9.978      1.092      0.278      -8.907      30.696
## x1             9.3559      3.448      2.714      0.008       2.514      16.198
## ==============================================================================
```

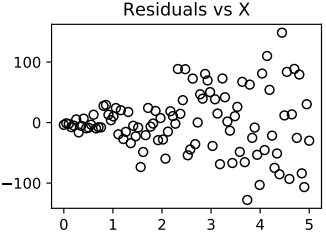












There are a number of methods to test for the presence of heteroskedasticity. . .

Goldfeld–Quandt Test

- It divides the dataset into two subsets. The subsets are specified so that the observations for which the explanatory variable takes the lowest values are in one subset, and the highest values - in the other subset.
- The subsets are not necessarily of equal size, nor do they contain all the observations between them.
- The test statistic used is the ratio of the mean square residual errors for the regressions on the two subsets.
- This test statistic corresponds to an F-test of equality of variances.
- The Goldfeld–Quandt test requires that data be **ordered** along a known explanatory variable, from lowest to highest.

If the error structure depends on an unknown variable or an unobserved variable the Goldfeld–Quandt test provides little guidance.
Also, error variance must be a **monotonic** function of the specified explanatory variable. For example, when faced with a quadratic function mapping the explanatory variable to error variance the Goldfeld–Quandt test may improperly accept the null hypothesis of homoskedastic errors.

Unfortunately the Goldfeld–Quandt test is not very robust to specification errors. The Goldfeld–Quandt test detects non-homoskedastic errors but cannot distinguish between heteroskedastic error structure and an underlying specification problem such as an incorrect functional form or an omitted variable.

Breusch–Pagan Test

- After estimating the linear regression $Y = \beta_0 + \beta_1 X + \epsilon$, calculate the model residuals $\widehat{\epsilon}_i$.
- The OLS assumptions state that the residual variance does not depend on the independent variables $\mathbb{V}\mathrm{ar}(\epsilon_i|\mathbf{X}) = \sigma^2_\epsilon$. If this assumptions is not true, then there may be a linear relationship between $\widehat{\epsilon}^2_i$ and X_i.

So, the Breush-Pagan test is the based on the following regression:

$$\widehat{\epsilon}^2_i = \gamma_0 + \gamma_1 X_i + v_i$$

The hypothesis tests is:

$$H_0 : \gamma_1 = 0 \text{ (residuals are homoskedastic)}$$
$$H_1 : \gamma_1 \neq 0 \text{ (residuals are heteroskedastic)}$$

It is a chi-squared test, where the test statistic:

$$LM = N \cdot R^2_{\widehat{\epsilon}}$$

is distributed as χ^2_1 under the null. Here $R^2_{\widehat{\epsilon}}$ is the R-square of the squared residual regression.

One weakness of the BP test is that it assumes that the heteroskedasticity is a **linear** relationship of the independent variables. If we fail to reject the null hypothesis, we still do not rule out the possibility of a non-linear relationship between the independent variables and the error variance.

White Test

- It is more generic than the **BP** test as it allows the independent variables to have a nonlinear effect on the error variance. For example, a combination of linear, quadratic and cross-products of the independent variables.
- It is a more commonly used test for homoskedasticity.

The test statistic is calculated the same way as in **BP** test:

$$LM = N \cdot R^2_{\hat{\epsilon}}$$

the difference from **BP** is that the squared residual model, from which we calculate $R^2_{\hat{\epsilon}}$, may be nonlinear.

A shortcoming of the White test is that it can lose its power if the model has many exogenous variables.

```
import statsmodels.stats.diagnostic as sm_diagnostic
# Goldfeld-Quandt Test
print(sm_diagnostic.het_goldfeldquandt(y = y,
          x = sm.add_constant(x), alternative = "two-sided"))
```

```
## (5.330019801721165, 4.3458162211165886e-08, 'two-sided')
```

```
# Breusch-Pagan Test
print(sm_diagnostic.het_breuschpagan(resid = mdl.resid,
          exog_het = sm.add_constant(x)))
```

```
## (27.15440005289671, 1.8783720306329005e-07, 36.53111796891305, 2.7232777548974932e-08)
```

```
# White Test
print(sm_diagnostic.het_white(resid = mdl.resid,
          exog = sm.add_constant(x)))
```

```
## (27.63658737233078, 9.972207411097485e-07, 18.522820288405395, 1.5369984549894296e-07)
```

- For **BP** and **White** tests - the first value is the *LM* statistic, the second value is the *p*-value of the LM statistic, the third value is the *F*-test statistic, the last value is the *p*-value of the *F* test.
- For the **GQ** test, the first value is the *F*-statistic, the second value is the associated *p*-value.

The LM test exaggerates the significance of results in small or moderately large samples. In this case, the F-statistic is preferable.

We see that in all cases the *p*-value is less than 0.05, so we reject the null hypothesis and conclude that the residuals **are** hetereoskedastic.

On the other hand, if we were to carry out these tests for a **correctly specified** model, like the one for the **simple linear regression**:

```
# Goldfeld-Quandt Test
print(sm_diagnostic.het_goldfeldquandt(y = data_lin["y"],
          x = sm.add_constant(data_lin["x"]), alternative = "two-sided"))
```

```
## (0.7299402182948976, 0.12090366054870887, 'two-sided')
```

```
# Breusch-Pagan Test
print(sm_diagnostic.het_breuschpagan(resid = mdl1.resid,
          exog_het = sm.add_constant(data_lin["x"])))
```

```
## (3.987557828302002, 0.04583745596968394, 4.027991495112321, 0.04611120607782332)
```

```
# White Test
print(sm_diagnostic.het_white(resid = mdl1.resid,
          exog = sm.add_constant(data_lin["x"])))
```

```
## (4.0785282350399354, 0.13012443194407053, 2.050490063862835, 0.13140921798003924)
```

We see that we do not reject the null hypothesis of homoskedastic residuals (except for the **BP** test in `Python`, where the p-value is close to 0.05, on the other hand, the remaining two tests do not reject the null).

There are also a number of additional heteroskedasticity tests. A discussion of their quality can be found here.

Residual Autocorrelation

If $\mathbb{C}\text{ov}(\epsilon_i, \epsilon_j) \neq 0$ for some $i \neq j$, then the errors are correlated. Autocorrelation is frequently encountered in time-series models.

Example

Assume that our model is defined as follows:

$$Y_t = \beta_0 + \beta_1 X_t + \epsilon_t$$
$$\epsilon_t = \rho\epsilon_{t-1} + u_t, \ |\rho| < 1, \ u_t \sim \mathcal{N}(0, \sigma^2)$$

Then we say that the model has **autocorrelated**, or **serially correlated** errors.
In this case, we have that:

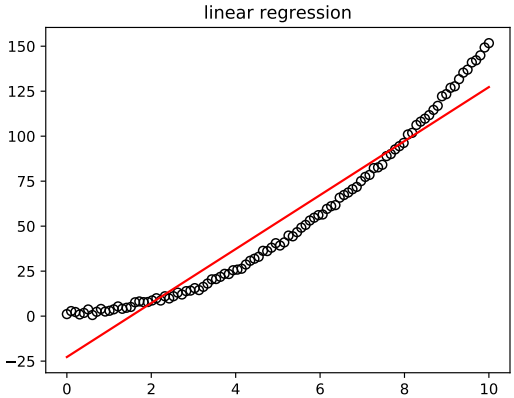
$$\mathbb{C}\text{ov}(\epsilon_t, \epsilon_{t-1}) = \mathbb{C}\text{ov}(\rho\epsilon_{t-1} + u_t, \epsilon_{t-1}) = \rho\mathbb{C}\text{ov}(\epsilon_{t-1}, \epsilon_{t-1}) = \rho\sigma^2 \neq 0$$

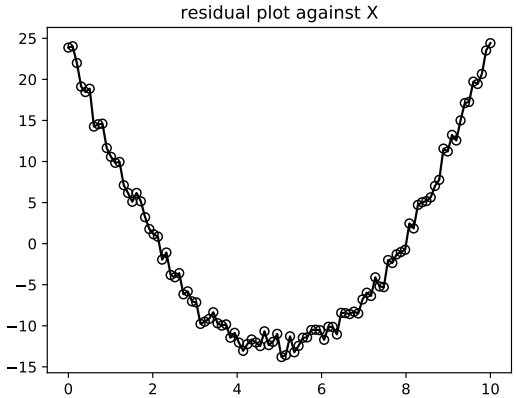
Estimating the coefficients via OLS and ignoring the violation will still result in unbiased and consistent OLS estimators. However, the estimators are inefficient and the variance of the regression coefficients will be **biased**.
On the other hand, autocorrelation in errors may be a result of a misspecified model.

Example

If we were to fit a linear model on a quadratic - we may get residuals, which appear to be correlated.

```
np.random.seed(123)
#
N = 100
beta_0 = 2
beta_1 = 1.5
#
x = np.linspace(start = 0, stop = 10, num = N)
e = np.random.normal(loc = 0, scale = 0.8, size = N)
y = beta_0 + beta_1 * (x**2) + e
#
lm_fit = sm.OLS(y, sm.add_constant(x)).fit()
```





There are a number of tests for the presence of autocorrelation. . .

Durbin–Watson Test

Tests the hypothesis:

H_0 : the errors are serially uncorrelated

H_1 : the errors follow a first order autoregressive process (i.e. autocorrelation at lag 1)

The test statistic:

$$d = \frac{\sum_{i=2}^{N} (\widehat{\epsilon}_i - \widehat{\epsilon}_{i-1})^2}{\sum_{i=1}^{N} \widehat{\epsilon}_i^2}$$

The value of d always lies between 0 and 4. $d = 2$ indicates no autocorrelation. If the Durbin–Watson statistic is not close to 2, there is evidence of a serial correlation.

Breusch-Godfrey Test

A more flexible test, covering autocorrelation of higher orders and applicable whether or not the regressors include lags of the dependent variable. Consider the following linear regression:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

We then estimate the model via OLS and fit the following model on the residuals $\widehat{\epsilon_i}$:

$$\widehat{\epsilon_i} = \alpha_0 + \alpha_1 X_i + \rho_1 \widehat{\epsilon}_{i-1} + \rho_2 \widehat{\epsilon}_{i-2} + ... + \rho_p \widehat{\epsilon}_{i-p} + u_t$$

and calculate its R^2 (R-squared), then testing the hypothesis:

$$H_0 : \rho_1 = \rho_2 = ... = \rho_p = 0$$
$$H_1 : \rho_j \neq 0 \text{ for some } j$$

Under the null hypothesis the test statistic:

$$LM = (N - p)R^2 \sim \chi^2_p$$

```
import statsmodels.stats.stattools as sm_tools
# Durbin-Watson Test
print(sm_tools.durbin_watson(lm_fit.resid))
```

```
## 0.018027275453585786
```

```
# Breusch-Godfrey Test
print(sm_diagnostic.acorr_breusch_godfrey(lm_fit, nlags = 2))
```

```
## (93.98337188826778, 3.9063405254180396e-21, 749.7890457680423, 2.5642011470769976e-59)
```

- For the Durbin–Watson Test, the DW statistic is returned. The test statistic equals 2 for no serial correlation. If it is closer to zero - we have evidence of positive correlation. If it is closer to 4, then we have more evidence of negative serial correlation.
- The Breusch-Godfrey Test returns the LM statistic with its corresponding p-value as well as the alternative test version with the F-statistic with its corresponding p-value.

In all test cases (because p-values are less than 0.05 and the Durbin-Watson test statistic is further from 2), we reject the null hypothesis of no serial correlation.

On the other hand, if we were to carry out these tests for a **correctly specified model**, like the one for the simple linear regression:

```
# Durbin-Watson Test
print(sm_tools.durbin_watson(mdl1.resid))
```

```
## 1.9377965734765383
```

```
# Breusch-Godfrey Test
print(sm_diagnostic.acorr_breusch_godfrey(mdl1, nlags = 2))
```

```
## (0.20396667467699192, 0.9030445986699857, 0.10004570053602482, 0.9048422424493952)
```

In this case the **DW** statistic is close to 2, and the test p-values are greater than 0.05, so we fail to reject the null hypothesis of no serial correlation.

Note There is also the **Ljung-Box Test** for testing the null hypothesis of no autocorrelation of residuals.

Residual Normality Testing

The normality requirement is necessary if we want to obtain the correct p-values and critical t-values when testing the hypothesis that $H_0 : \beta_j = c$, especially for significance testing, with $c = 0$. Assume that we want to test whether our residuals $z_1, ..., z_N$ come from a normal distribution. The hypothesis can be stated as:

$$H_0 : \text{residuals follow a normal distribution}$$
$$H_1 : \text{residuals do not follow a normal distribution}$$

There are a number of normality tests...

Anderson-Darling Test.

The test statistic is calculated as:

$$A^2 = -N - \sum_{i1}^{N} \frac{2i-1}{N} \left[\log(F(z_{(i)}) + \log\left(1 - F(z_{(N+1-i)})\right)\right]$$

where $z_{(i)}$ are the **ordered data** and $F(\cdot)$ is the cumulative distribution function (cdf) of the distribution being tested (for the univariate regression residuals - we are usually interested in testing for the normal distribution). The test statistic is compared against the critical values from the normal distribution. Empirical testing indicates that the Anderson–Darling test is not quite as good as Shapiro-Wilk, but is better than other tests.

Shapiro-Wilk Test.

The test statistic is:

$$W = \frac{\left(\sum_{i=1}^{N} a_i z_{(i)}\right)^2}{\sum_{i=1}^{N}(z_i - \overline{z})^2}$$

where $z_{(i)}$ is the i-th **smallest value in the sample** (i.e. the data are ordered). a_i values are calculated using means, variances and covariances of $z_{(i)}$. W is compared against tabulated values of this statistic's distribution. Small values of W will lead to the rejection of the null hypothesis. Monte Carlo simulation has found that Shapiro–Wilk has the best power for a given significance, followed closely by Anderson–Darling when comparing the Shapiro–Wilk, Kolmogorov–Smirnov, Lilliefors and Anderson–Darling tests.

Kolmogorov-Smirnov Test.

The test statistic is given by:

$$D = \max\{D^+; D^-\}$$

where:

$$D^+ = \max_i \left(\frac{i}{N} - F(z_{(i)}) \right)$$

$$D^- = \max_i \left(F(z_{(i)}) - \frac{i-1}{N} \right)$$

where $F(\cdot)$ is the theoretical cdf of the distribution being tested (for the univariate regression residuals - we are usually interested in testing for the normal distribution). The **Lilliefors Test** is based on the Komogorov-Smirnov Test as a special case of this for the normal distribution.. For the normal distribution case, the test statistic is compared against the critical values from a normal distribution in order to determine the p-value.

Cramer–von Mises Test

Can be though of as an alternative to the Kolmogorov–Smirnov test. The test statistic:

$$W = N\omega^2 = \frac{1}{12N} + \sum_{i=1}^{N} \left[\frac{2i-1}{2N} - F(z_{(i)}) \right]^2$$

If this value is larger than the tabulated value, then the hypothesis that the data came from the distribution F can be rejected.

Jarque–Bera Test

This test is valid for large samples. The statistic is calculated as:

$$JB = \frac{N-k+1}{6}\left(S^2 + \frac{(C-3)^2}{4}\right)$$

where

$$S = \frac{\frac{1}{N}\sum_{i=1}^{N}(z_i - \overline{z})^3}{\left(\frac{1}{N}\sum_{i=1}^{N}(z_i - \overline{z})^2\right)^{3/2}} = \frac{\widehat{\mu}_3}{\widehat{\sigma}^3}$$

$$C = \frac{\frac{1}{N}\sum_{i=1}^{N}(z_i - \overline{z})^4}{\left(\frac{1}{N}\sum_{i=1}^{N}(z_i - \overline{z})^2\right)^{2}} = \frac{\widehat{\mu}_4}{\widehat{\sigma}^4}$$

N is the sample size, S is the skewness and C is kurtosis and k is the number of regressors (i.e. the number of different independent variables X, with $k = 1$ outside a regression context). If the data comes from a normal distribution, then the JB statistic has a chi-squared distribution with **two degrees of freedom**, χ^2_2.

Chi-squared (Goodness-Of-Fit) Test.

The chi-square goodness-of-fit test can be applied to discrete distributions such as the binomial and the Poisson, while the Kolmogorov-Smirnov and Anderson-Darling tests are restricted to continuous distributions.

This is not a restriction per say, since for non-binned data you can simply calculate a histogram before generating the chi-square test. However, the value of the chi-square test statistic are dependent on how the data is binned. Another disadvantage of the chi-square test is that it requires a sufficient sample size in order for the chi-square approximation to be valid.

We will carry out the normality tests on the log-linear DGP, with an incorrectly specified linear model:

```
# May need to install through terminal: pip install scikit-gof
import skgof as skgof
# Anderson-Darling Test
print(sm_diagnostic.normal_ad(x = mdl3.resid))
```

```
## (2.742775345091559, 6.263145991939627e-07)
```

```
# Shapiro-Wilk Test
print(stats.shapiro(x = mdl3.resid))
```

```
## (0.9430360198020935, 4.2506789554863644e-07)
```

```
# Kolmogorov-Smirnov Test
print(sm_diagnostic.kstest_normal(x = mdl3.resid, dist = "norm")) #statistic and p-value
```

```
## (0.08586356720817684, 0.0010846493406071833)
```

```
# Cramer-von Mises test
print(skgof.cvm_test(data = mdl3.resid,
                     dist = stats.norm(0, np.sqrt(np.var(mdl3.resid)))))
```

```
## GofResult(statistic=0.39458042141994304, pvalue=0.07458108247027562)
```

```
# Jarque-Bera Test
print(sm_tools.jarque_bera(mdl3.resid)) #JB statistic, pvalue, skew and kurtosis.
```

```
## (27.00245322122904, 1.3692784843495163e-06, 0.8659632801063863, 3.490637112922614)
```

Note that the Jarque-Bera tests in `R` and `Python` in these packages do not allow to control for the fact that we are carrying out the tests on the residuals. In other words, it assumes that $k = 1$. In this case the p-value is less than 0.05 for most tests, so we reject the null hypothesis and conclude that the residuals are not normally distributed.

For a **correctly specified** log-linear model:

```
# Anderson-Darling Test
print(sm_diagnostic.normal_ad(x = mdl3_correct.resid))
```

```
## (0.1576307896975777, 0.9518802524386675)
```

```
# Shapiro-Wilk Test
print(stats.shapiro(x = mdl3_correct.resid))
```

```
## (0.9960342049598694, 0.8864037990570068)
```

```
# Kolmogorov-Smirnov Test
print(sm_diagnostic.kstest_normal(x = mdl3_correct.resid, dist = "norm")) #statistic and p-value
```

```
## (0.031206131899122358, 0.2)
```

```
# Cramer-von Mises test
print(skgof.cvm_test(data = mdl3_correct.resid,
                     dist = stats.norm(0, np.sqrt(np.var(mdl3_correct.resid)))))
```

```
## GofResult(statistic=0.02066788690314537, pvalue=0.996414601640607)
```

```
# Jarque-Bera Test
print(sm_tools.jarque_bera(mdl3_correct.resid)) #JB statistic, pvalue, skew and kurtosis.
```

```
## (0.4627520589444065, 0.793441052712381, -0.11645049039934306, 2.9641199189474325)
```

In this case, the p-values for the tests are greater than 0.05, so we do not reject the null hypothesis of normality.

The more tests we carry out the more certain we can be, whether the residuals are (not) from a normal distribution.
For now, focus on at least one test from each category - namely:

- **Breusch–Pagan Test** for homoskedasticity,
- **Durbin-Watson Test** for autocorrelation,
- **Shapiro-Wilk Test** for normality.

Standardized Residuals

When we compare residuals for different observations, we want to take into account that their variances may be different (as we have shown in eq. @ref(eq:residVar)). One way to account for this is to divide the residuals by an estimate the **residuals standard deviation**. This results in calculating the **standardized** residuals:

$$s_i = \frac{\widehat{\epsilon_i}}{\widehat{\sigma}\sqrt{1 - h_{ii}}}$$

where h_{ii} is the i-th diagonal element of $\mathbf{H}$. Standardized residuals are useful in detecting outliers. Generally, any observation with a standardized residual greater than 2 in absolute value should be examined more closely.

Lecture Summary

We have :

- re-examined hypothesis testing;
- learned about predictions and their intervals. We have compared them with confidence intervals.
- examined various measures of model goodness of fit in terms of R^2. We have looked at the various drawbacks of such measures (see the counter examples).
- analysed regression diagnostic tests - what kind of problems arise; what tests are available; how to carry them out.

Examples using empirical data

From the Lecture notes `Ch. 3.10` continue with the dataset(-s) that you have used from the previous exercise set and do the tasks from `Exercise Set 3` from `Ch 3.10`. See `Ch. 3.11` for an example.

Next Lecture

- Subsampling + Review of theory + Some `Python` code summarization;