Applied Statistical Methods

Tests of Means of Numerical Data- Part III

Xuemao Zhang
East Stroudsburg University

March 3, 2023

Outline

Kruskal-Wallis Test

- The Kruskal-Wallis H Test is a nonparametric procedure that can be used to compare more than two populations in a completely randomized design.
- All $n = n_1 + n_2 + \ldots + n_k$ measurements are jointly ranked.
- We use the sums of the ranks of the k samples $(k > 2)$ to compare the distributions.
- Example: IQ scores from a sample of subjects with low, medium, and high lead exposure (IQScores.csv). Test the claim that the three sample medians come from populations with medians that are all equal.

Kruskal-Wallis Test

- https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kruskal.html

```
import pandas as pd
from scipy import stats
IQ = pd.read_csv("../data/IQScores.csv")
score1=IQ[IQ.LeadLevel=='L'].Scores
score2=IQ[IQ.LeadLevel=='M'].Scores
score3=IQ[IQ.LeadLevel=='H'].Scores
stats.kruskal(score1,score2,score3)
```

```
## KruskalResult(statistic=0.7031083481349935, pvalue=0.7035937323951664)
```

- R code

```
library(readr)
IQ = read_csv("../data/IQScores.csv")
str(IQ)
tapply(IQ$Scores, IQ$LeadLevel, median, na.rm=TRUE)
kruskal.test(Scores ~ LeadLevel, data=IQ)
```

Friedman Rank-Sum Test

- The Friedman rank-sum test is a nonparametric procedure that can be used to compare more than two population medians in a randomized complete block design. The procedure involves ranking each row(block) together, then considering the values of ranks by treatments(columns).
  - https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.friedmanchisquare.html

```
Soil = pd.read_csv("../data/Soil.csv")
gA=Soil[Soil.soil=='A'].growth
gB=Soil[Soil.soil=='B'].growth
gC=Soil[Soil.soil=='C'].growth
stats.friedmanchisquare(gA,gB,gC)
```

```
## FriedmanchisquareResult(statistic=6.533333333333333, pvalue=0.0381333265
```

- R code

```
library(readr)
Soil = read_csv("../data/Soil.csv")
friedman.test(formula=growth~soil|location, data=Soil)
```

Multiple Comparisons

After conducting ANOVA, there are several informal methods for determining which means are different:

- Construct boxplots of the different samples to see if one or more of them is very different from the others.
- Construct confidence interval estimates of the means for the different samples, then compare those confidence intervals to see if one or more of them does not overlap with the others (pairwise comparison) or conduct pairwise hypotheses. The method is called LSD (Least Significant Difference).
    - LSD problem: In hypotheses test problems involving a single null hypothesis H_0 the statistical tests are often chosen to control the Type I error rate of incorrectly rejecting H_0 at a pre-specified significance level α. In general, when testing m null hypotheses using independent test statistics, the probability of committing at least one Type I error is $1-(1-\alpha)^m$.

Multiple Comparisons

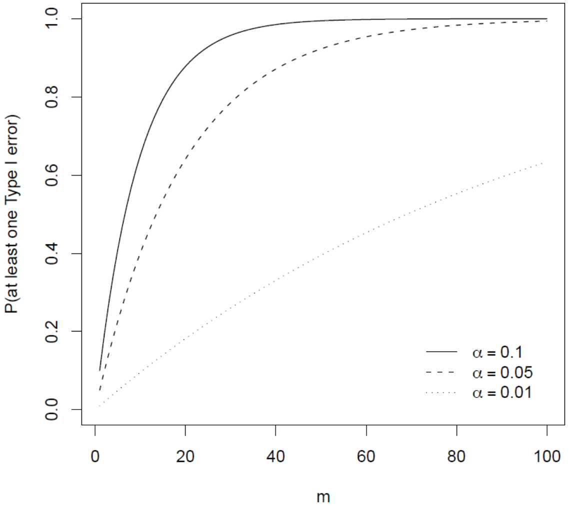


Figure 1: Probability of committing at least one Type I error for diferent significance levels α and number of hypotheses *m*.

Bonferroni test

- We can run t-test on all pairs of hypothesis tests, calculate the p-values and apply a p-value correction for multiple testing problems.
- The simplest - and at the same time quite conservative - approach is to which compares the unadjusted p-values $p_1, \ldots, p_m$ with the common threshold α/m, where m is the number of hypotheses under investigation.
- Equivalently, a null hypothesis $H_{0i} : i \in \{1, 2, \ldots, m\}$ is rejected, if the adjusted p-value $q_i = \min\{mp_i, 1\}$. Here, the minimum is used to ensure that the resulting adjusted p-value q_i is not larger than 1.
- Example: If you compare four comparisons, you check for significance not at $\alpha = 0.05$, bu at $\alpha/4 = 0.0125$.

Bonferroni test

- `statsmodels.stats.multitest.multipletests()` function can be used to adjust the p-values of pairwise tests.
  - For available **adjusted methods**, see https://www.statsmodels.org/dev/generated/statsmodels.stats.multitest.multipletests.html
- To calculate the adjusted p-values in the previous example, we first define a vector containing the unadjusted p-values and subsequently call the `p.adjust` function.
- For example,if we obtain 3 p-values for all 3 pairwise tests, then

```
from statsmodels.stats.multitest import multipletests
import numpy as np
p=np.array([0.01, 0.015, 0.005])
multipletests(p, method='bonferroni')
```

```
## (array([ True,  True,  True]), array([0.03 , 0.045, 0.015]), 0.016952427
```

```
# R code:
p = c(0.01, 0.015, 0.005)
p.adjust(p, "bonferroni")
```

Bonferroni test

- To conduct multiple comparisons by Bonferroni's method, method `t_test_pairwise()` can be used.
  - https://www.statsmodels.org/stable/generated/statsmodels.base.distributed_estimation.DistributedResults.t_test_pairwise.html

```
import pandas as pd
from statsmodels.formula.api import ols
Breakfast=pd.DataFrame({"resp":[8,7,9,13,14,16,12,17,10,12,16,15],
'Trt':["1"]*4+["2"]*4+["3"]*4} )
Model1=ols('resp ~ Trt', data=Breakfast).fit()
# type(Model1)
ptest=Model1.t_test_pairwise("Trt", method="bonferroni")
print(ptest.result_frame)
```

```
##      coef   std err  ...  pvalue-bonferroni  reject-bonferroni
## 2-1   5.5  1.798919  ...           0.040887               True
## 3-1   4.0  1.798919  ...           0.159762              False
## 3-2  -1.5  1.798919  ...           1.000000              False
##
## [3 rows x 8 columns]
```

Bonferroni test

- To conduct multiple comparisons by Bonferroni's method, the function `pairwise.t.test()` in R can be used.

```
resp=c(8,7,9,13,14,16,12,17,10,12,16,15)
Trt=c(rep(1,4),rep(2,4),rep(3,4))
Breakfast=as.data.frame(cbind(resp, Trt))
Breakfast$Trt =factor(Breakfast$Trt)
pairwise.t.test(x=Breakfast$resp, g=Breakfast$Trt, p.adj = "bonf")
```

Holm procedure

- Holm ("A simple sequentially rejective multiple test procedure", 1979) introduced a multiple comparison procedure, which uniformly improves the Bonferroni approach.
- The Holm procedure is a step-down procedure, which basically consists of repeatedly applying Bonferroni's method while testing the hypotheses in a data-dependent order.
- Let $p_{(1)}, \dots, p_{(m)}$ denote the **ordered** unadjusted p-values with associated hypotheses $H_{(01)}, \dots, H_{(0m)}$. Then, $H_{(0i)}$ is rejected if $p_{(j)} \leq \alpha/(n-j+1), j = 1, \dots, i$. That is, $H_{(0i)}$ is rejected if $p_{(i)} \leq \alpha/(m-i+1)$ and all hypotheses $H_{(0j)}$ preceding $H_{(0i)}$ are also rejected.

Holm procedure

- The Holm procedure can be described by the following sequentially rejective test procedure:
  - Start testing the null hypothesis $H_{(01)}$ associated with the smallest p-value $p_{(1)}$. If $p_{(1)} > \alpha/m$, the procedure stops and no hypothesis is rejected. Otherwise, $H_{(01)}$ is rejected and the procedure continues testing $H_{(02)}$ at the larger significance level $\alpha/(m-1)$. These steps are repeated until either the first non-rejection occurs or all null hypotheses $H_{(01)}, \ldots, H_{(0m)}$ are rejected.

```
from statsmodels.stats.multitest import multipletests
Model1=ols('resp ~ Trt', data=Breakfast).fit()
ttests=Model1.t_test_pairwise("Trt", method="holm")
ttests.result_frame
```

```
##      coef   std err         t  ...  Conf. Int. Upp.  pvalue-holm  reject
## 2-1   5.5  1.798919  3.057391  ...         9.569438     0.040887
## 3-1   4.0  1.798919  2.223557  ...         8.069438     0.106508
## 3-2  -1.5  1.798919 -0.833834  ...         2.569438     0.425951
##
## [3 rows x 8 columns]
```

- R code

```
pairwise.t.test(x=Breakfast$resp, g=Breakfast$Trt, p.adj = "holm")
```

Tukey's test

- Tukey's test, also referred to as the Tukey HSD (Honest Significance Difference) test, controls for the Type I error rate across multiple comparisons.
- Tukey's test is based on the studentized range. In essence, the Tukey test takes the maximum over the absolute values of all pairwise test statistics.
- For testing the difference between group i and j, the Tukey's test statistic is

$$t_{ij} = \frac{\bar{y}_i - \bar{y}_j}{s\sqrt{\frac{2}{n}}},$$

  where, s the pooled standard deviation and n the common sample size. Each test statistic t_{ij} is univariate t distributed. The vector of the test statistics follows a multivariate t distribution. Until the emergence of modern computing facilities, the exact calculation of critical values for the Tukey test was only possible for limited cases.

Tukey's test

- Tukey's test can be used as post-hoc analysis to test between whicn two group means there is a significant difference after rejecting the null hypothesis that all group means are equal.

$$HSD = q_{k,df,\alpha}\sqrt{\frac{MS_E}{n}}$$

where k is the number of groups, df is the error degrees of freedom, and $q_{k,df,\alpha}$ is determined by the studentized range (q) distributiondistribution of

$$q_k = \frac{\max\{\bar{y}_i, \dots, \bar{y}_k\} - \min\{\bar{y}_i, \dots, \bar{y}_k\}}{\sqrt{MS_E/n}}$$

If the absolute difference between two group sample means is larger than the HSD, then we say the difference is significant.

- For unbalanced designs, we just change the statistics accordingly.

Tukey's test

- Function `scipy.stats.tukey_hsd()`: can be used for Tukey test https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.tukey_hsd.html

```
from scipy.stats import tukey_hsd
group1=Breakfast[Breakfast.Trt=='1'].resp
group2=Breakfast[Breakfast.Trt=='2'].resp
group3=Breakfast[Breakfast.Trt=='3'].resp
tukeyresults2=tukey_hsd(group1,group2,group3)
print(tukeyresults2)
```

Tukey's test

- The best way is to use `statsmodels.stats.multicomp.pairwise_tukeyhsd()`: https://www.statsmodels.org/dev/generated/statsmodels.stats.multicomp.pairwise_tukeyhsd.html

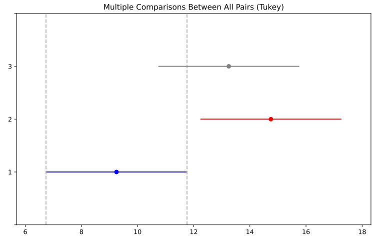
```
from statsmodels.stats.multicomp import (pairwise_tukeyhsd, MultiComparison)
tukeyresults1=pairwise_tukeyhsd(endog=Breakfast['resp'],
groups=Breakfast['Trt'],alpha=0.05)
print(tukeyresults1)
```

```
## Multiple Comparison of Means - Tukey HSD, FWER=0.05
## ====================================================
## group1 group2 meandiff p-adj   lower   upper  reject
## ----------------------------------------------------
##      1      2      5.5 0.0331  0.4774 10.5226   True
##      1      3      4.0 0.1202 -1.0226  9.0226  False
##      2      3     -1.5 0.6926 -6.5226  3.5226  False
## ----------------------------------------------------
```

Tukey's test

- Any two pairs can be compared for significance by looking for overlap using method `TukeyHSDResults.plot_simultaneous()`.
    - https://www.statsmodels.org/dev/generated/statsmodels.sandbox.stats.multicomp.TukeyHSDResults.plot_simultaneous.html
    - `comparison_name`: if provided, plot_intervals will color code all groups that are significantly different from the comparison_name red, and will color code insignificant groups gray. Otherwise, all intervals will just be plotted in black.

```
import matplotlib.pyplot as plt
tukeyresults1.plot_simultaneous(comparison_name="1")
plt.show()
```



Tukey's test

- Or use MultiComparison

```
comp=MultiComparison(Breakfast['resp'],Breakfast['Trt'])
#type(comp)
post_hoc = comp.tukeyhsd()
print(post_hoc.summary())
```

```
## Multiple Comparison of Means - Tukey HSD, FWER=0.05
## ====================================================
## group1 group2 meandiff p-adj   lower   upper  reject
## ----------------------------------------------------
##      1      2      5.5 0.0331  0.4774 10.5226   True
##      1      3      4.0 0.1202 -1.0226  9.0226  False
##      2      3     -1.5 0.6926 -6.5226  3.5226  False
## ----------------------------------------------------
```

Tukey's test

- R code

```
Model1=lm(resp ~ Trt, data= Breakfast)
TukeyHSD(aov(Model1), conf.level = 0.95)
# The package `agricolae` can help us visualize the multiple compar
library(agricolae)
plot(TukeyHSD(aov(Model1), conf.level = 0.95),las=1, col = "red")
# Visualization of the grouping:
compmeans=HSD.test(aov(Model1),"Trt",alpha=0.05,group=TRUE);
plot(compmeans, main="Multiple comparisons")
box()
```

Many-to-one comparisons: Dunnett test

- Dunnett test is the standard method for the the classical many-to-one problem of comparing several groups with a common control group. Suppose there are $m+1$ treatment and let μ_0 be the mean of the control group. Then we are testing $H_{0i} : \mu_0 = \mu_i, i = 1, \ldots, m$ against one of the three alternatives $(>, <, \neq)$.
    - Rejecting any of the null hypotheses thus ensures that at least one of the alternative is supported at a given confidence level $1-\alpha$, if suitable multiple comparison procedures are employed.
- The one-sided Dunnett test takes the minimum (or the maximum, depending on the sideness of the test problem) of the m, say, pairwise t tests

$$t_i = \frac{\bar{y}_i - \bar{y}_0}{s\sqrt{\frac{1}{n_i} + \frac{1}{n_0}}}, i = 1, \ldots, m.$$

Many-to-one comparisons: Dunnett test

- Each test statistic t_i is univariate t distributed. The vector of test statistics $t = (t_1, \ldots, t_m)$ follows an m-variate t distribution with degrees of freedom $\sum_{i=0}^{m} n_i - (m+1)$ (and a correlation matrix).
    - The `multcomp` package in R, can be used to calculate adjusted p-values or critical values.
- Example: A company developed specialized heating blankets designed to help the body heat following a surgical procedure. Four types of blankets b_0, b_1, b_2 and b_3 were tested on surgical patients to assess recovery times. The blanket b_0 was a standard blanket already in use at various hospitals. The primary outcome of interest was recovery time in minutes of patients allocated randomly to one of the four treatments. Lower recovery times would indicate a better treatment effect.

Many-to-one comparisons: Dunnett test

- Python statistical ecosystem is comprised of multiple packages. However, it still has numerous gaps and is surpassed by R packages and capabilities.
- Package `scikit-posthocs` attempts to improve Python statistical capabilities by offering a lot of parametric and nonparametric post hoc tests along with outliers detection and basic plotting methods.
  - https://scikit-posthocs.readthedocs.io/en/latest/intro/
  - https://scikit-posthocs.readthedocs.io/en/latest/generated/scikit_posthocs.posthoc_dunn

```
pip install scikit-posthocs
```

```
import scikit_posthocs as sp
import pandas as pd
recovery=pd.read_csv("../data/recovery.csv")
sp.posthoc_dunn(recovery,val_col='minutes', group_col='blanket',
p_adjust = 'holm')
```

```
##            b0         b1         b2         b3
## b0  1.000000  0.600997  0.012189  0.434658
## b1  0.600997  1.000000  0.434658  0.882987
## b2  0.012189  0.434658  1.000000  0.156105
## b3  0.434658  0.882987  0.156105  1.000000
```

Many-to-one comparisons: Dunnett test

- R code

```r
# The `glht` function from package `multcomp` takes the fitted response
#model to perform the multiple comparisons
library(multcomp)
data(recovery)
siglevel= 0.05
recovery$blanket <- relevel(recovery$blanket, ref = "b0")
# b0 is set as reference level
recovery.aov <- aov(minutes ~ blanket, data = recovery)
recovery.mc <- glht(recovery.aov,linfct = mcp(blanket="Dunnett"),
                    alternative = "two.sided")
#the mcp function for the linfct argument is used
#to specify the comparisons type
summary(recovery.mc)
```

```r
# Alternatively, we can use the `confint` method associated
# with the `glht` function.
recovery.ci <- confint(recovery.mc, level = 1-siglevel)
recovery.ci$confint #confidence intervals
recovery.ci
```

```r
# In addition,we can display the confidence intervals graphically.
plot(recovery.ci, main = "", ylim = c(0.5, 3.5), xlab = "Minutes")
```

Analysis of Covariance - Introduction

- ANCOVA (Analysis of Covariance) is really "ANOVA with covariates" or, more simply, a combination of ANOVA and regression used when you have some categorical factors and some quantitative predictors.
- The predictors (X variables on which to perform regression) are called "covariates" in this context.
- The idea is that often these covariates are not necessarily of primary interest, but still their inclusion in the model will help explain more of the response, and hence reduce the error variance.

Analysis of Covariance - Introduction

- Example: Consider a study performed to determine if there is a difference in the strength of monofilament fiber produced by three different machines.

Table 1: Breaking Strength Data(y=strength in pounds and x= diameter in 10^{-3} in.)

Machine 1		Machine 2		Machine 3	
y	x	y	x	y	x
36	20	40	22	35	21
41	25	48	28	37	23
39	24	39	22	42	26
42	25	45	30	34	21
49	32	44	28	32	15

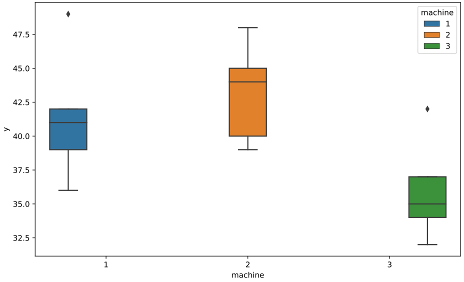
Analysis of Covariance - Introduction

```
import pandas as pd
import numpy as np
y=[36,41,39,42,49,40,48,39,45,44,35,37,42,34,32]
x=[20,25,24,25,32,22,28,22,30,28,21,23,26,21,15]
machine=['1']*5+['2']*5+['3']*5
dataset=pd.DataFrame({'x':x, 'y':y, 'machine':machine})
dataset.groupby(['machine']).agg([np.mean, np.std])
```

```
##             x                y
##          mean       std  mean       std
## machine
## 1        25.2  4.324350  41.4  4.827007
## 2        26.0  3.741657  43.2  3.701351
## 3        21.2  4.024922  36.0  3.807887
```

Analysis of Covariance - Introduction

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.boxplot(data=dataset, x="machine", y='y', hue="machine")
plt.show()
```



Analysis of Covariance - Introduction

- R code

```
library(dplyr)
library(ggplot2)
y=c(36,41,39,42,49,40,48,39,45,44,35,37,42,34,32);
x=c(20,25,24,25,32,22,28,22,30,28,21,23,26,21,15);
machine=c(rep(1,5),rep(2,5),rep(3,5));
dataset=as.data.frame(cbind(x,y,machine));
dataset$machine = as.factor(dataset$machine);
dataset%>%group_by(machine)%>%summarise(meany=mean(y), meanx=mean(x))
```

```
## # A tibble: 3 x 3
##   machine meany meanx
##   <fct>   <dbl> <dbl>
## 1 1        41.4  25.2
## 2 2        43.2  26
## 3 3        36    21.2
```

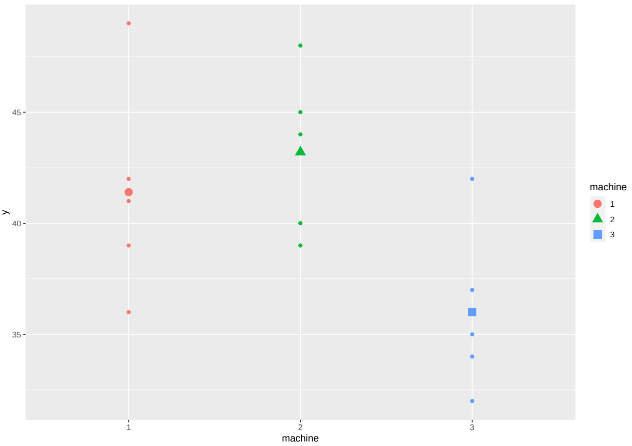
```
library(ggplot2)
ggplot(data=dataset, aes(x=machine,y=y, color=machine))+
  geom_boxplot(aes(group=machine))+
  geom_point()
```

Analysis of Covariance - Introduction

Several statistical models can be proposed:

- Model 1: no covariates are involved

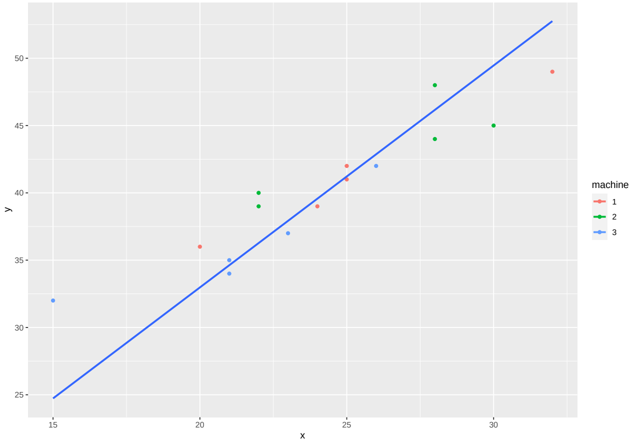
$$H_1 : Y_{ij} = \mu_i + \varepsilon_{ij}.$$



Analysis of Covariance - Introduction

- Model 2:

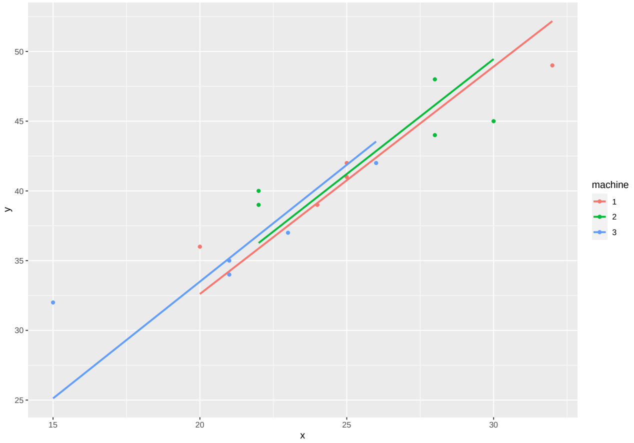
$$H_2 : Y_{ij} = \mu + \beta x_{ij} + \varepsilon_{ij}.$$



Analysis of Covariance - Introduction

- Model 3:

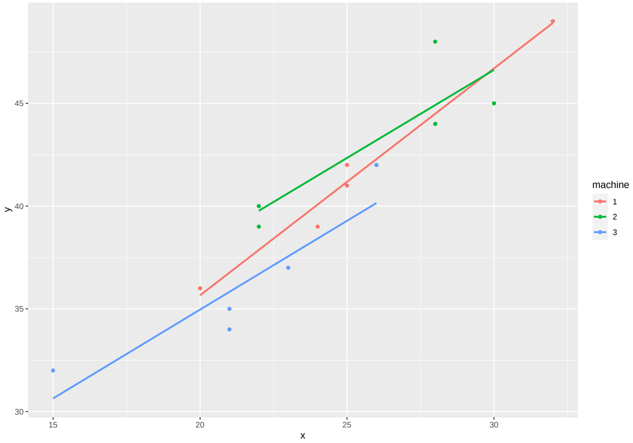
$$H_3 : Y_{ij} = \mu_i + \beta x_{ij} + \varepsilon_{ij}.$$



Analysis of Covariance - Introduction

- Model 4:

$$H_4 : Y_{ij} = \mu_i + \beta_i x_{ij} + \varepsilon_{ij}.$$



Analysis of Covariance

Table 2: Data from 1-way design with covariates

Trt 1		$\cdots$		Trt a	
y	x			y	x
y_{11}	x_{11}	$\cdots$	$\cdots$	y_{a1}	x_{a1}
y_{12}	x_{12}	$\cdots$	$\cdots$	y_{a2}	x_{a2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
y_{1n}	x_{1n}	$\cdots$	$\cdots$	y_{an}	x_{an}

Analysis of Covariance

- ANCOVA is assuming model H_3 and test for H_2
- ANCOVA model for a balanced design

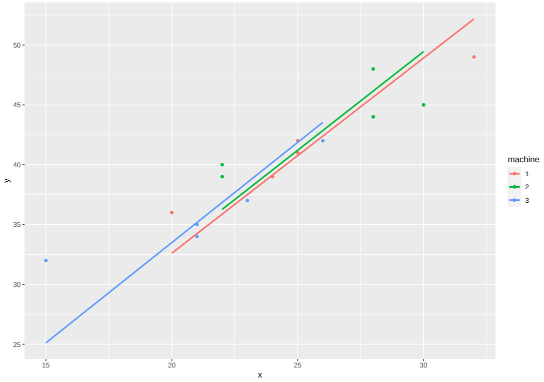
$$Y_{ij} = \mu + \tau_i + \beta(x_{ij} - \bar{x}_{\cdot\cdot}) + \varepsilon_{ij}, i = 1, \ldots, a; j = 1, \ldots, n.$$

Table 3: Analysis of Covariance Table

Source	df	SS	MS	F
Regression	1	SS_{cov}	SS_{cov}	MS_{cov}/MSE
Treatments	$a-1$	SS_T	$SS_T/(a-1)$	MS_T/MSE
Error	$a(n-1)-1$	SSE	$\frac{SSE}{a(n-1)-1}$	
Total	$an-1$	SS_{total}		

Analysis of Covariance

- Parallel regression lines
  - Given data, we can plot regression fit without intercepts to compare their slopes
  - Unfortunately, `seaborn` cannot plot regression lines without intercepts. You my try `plotnine` library to use `ggplot`.



Analysis of Covariance

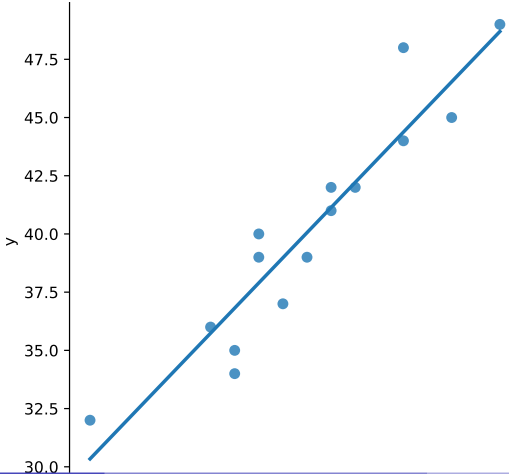
- Parallel regression lines
- R code

```
ggplot(data=dataset, aes(x=x,y=y, color=machine))+
  geom_point()+
  geom_smooth(method=lm, formula=y~0+x, se=FALSE)
```

Analysis of Covariance

- Plotting a single regression line

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.lmplot(x = "x", y = "y",ci = None,data = dataset)
```



Analysis of Covariance

- Plotting a single regression line
- R code

```
ggplot(data=dataset, aes(x=x,y=y, color=machine))+
  geom_point()+
  geom_smooth(method=lm, se=FALSE, aes(group=1));
```

Analysis of Covariance

- Model fit

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
lmfit = ols('y~ x+ machine', data=dataset).fit()
```

- R code

```
lmfit <- lm(y~ x+ machine, data=dataset)
summary(lmfit)
anova(lmfit)
lmfit$coefficients
confint(lmfit)
```

Analysis of Covariance

- Model fit

```
print(lmfit.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:                      y   R-squared:
## Model:                            OLS   Adj. R-squared:
## Method:                 Least Squares   F-statistic:
## Date:                Sat, 25 Feb 2023   Prob (F-statistic):
## Time:                        11:51:14   Log-Likelihood:
## No. Observations:                  15   AIC:
## Df Residuals:                      11   BIC:
## Df Model:                           3
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.
## ------------------------------------------------------------------------------
## Intercept     17.3595      2.961      5.862      0.000      10.
## machine[T.2]   1.0368      1.013      1.024      0.328      -1.
## machine[T.3]  -1.5840      1.107     -1.431      0.180      -4.
```

Analysis of Covariance

- ANOVA table

```
anova_table=sm.stats.anova_lm(lmfit, typ=2)
print(anova_table)
```

```
##                 sum_sq    df          F    PR(>F)
## machine      13.283851   2.0   2.610643  0.118084
## x           178.014110   1.0  69.969375  0.000004
## Residual     27.985890  11.0        NaN       NaN
```

Analysis of Covariance

- Estimate of the coefficients

```
print(lmfit.params)
```

```
## Intercept        17.359509
## machine[T.2]      1.036810
## machine[T.3]     -1.584049
## x                 0.953988
## dtype: float64
```

- Confidence intervals

```
lmfit.conf_int(alpha=0.05)
```

```
##                         0          1
## Intercept       10.841914  23.877105
## machine[T.2]    -1.192597   3.266217
## machine[T.3]    -4.020870   0.852771
## x                0.702969   1.205006
```

Analysis of Covariance

- Adjusted means: group means adjusted for the effect of the covariate.
- The following output shows the adjusted means

```
adjusted_means = lmfit.get_prediction().summary_frame()
adjusted_means=adjusted_means.join(dataset)
#adjusted_means
grouped_means = adjusted_means.groupby('machine')['mean'].mean()
grouped_means
```

```
## machine
## 1    41.4
## 2    43.2
## 3    36.0
## Name: mean, dtype: float64
```

```
#We use the `effect(`) function in the `effects` package
library(effects)
adjustedMeans<-effect("machine", lmfit, se=TRUE);
# # se=TRUE: show standard errors
summary(adjustedMeans)
```

Analysis of Covariance

- Post hoc tests
  - We conduct the test if "H_0 : All group means (adjusted) are equal" is rejected

```
## Perform post-hoc Tukey test with both covariates
from statsmodels.sandbox.stats.multicomp import MultiComparison
mc = MultiComparison(dataset['y'], dataset['machine'])
#covariate is not included!
mcresult = mc.tukeyhsd(alpha=0.05)
print(mcresult.summary())
```

```
## Multiple Comparison of Means - Tukey HSD, FWER=0.05
## ====================================================
## group1 group2 meandiff p-adj   lower  upper  reject
## ----------------------------------------------------
##      1      2      1.8 0.7754  -5.191  8.791  False
##      1      3     -5.4 0.1402 -12.391  1.591  False
##      2      3     -7.2 0.0434 -14.191 -0.209   True
## ----------------------------------------------------
```

- R code

```
#We use the `glht()` function in the `multcomp` package
library(multcomp)
postHocs<-glht(lmfit, linfct = mcp(machine = "Tukey"))
summary(postHocs)
confint(postHocs,level = 0.95)
```

Analysis of Covariance

- Confidence intervals of post-hoc

```
print(mcresult.confint)
```

```
## [[ -5.19095672   8.79095672]
##  [-12.39095672   1.59095672]
##  [-14.19095672  -0.20904328]]
```

- A relatively new python package Pingouin for ANCOVA
  - https://pingouin-stats.org/build/html/api.html

```
import pingouin as pg
pg.ancova(data=dataset, dv='y', between='machine', covar='x')
```

```
##      Source          SS  DF          F     p-unc       np2
## 0   machine   13.283851   2   2.610643  0.118084  0.321879
## 1         x  178.014110   1  69.969375  0.000004  0.864146
## 2  Residual   27.985890  11        NaN       NaN       NaN
```

License