MTH 4230 R Notes 1

1 Correlation

1.1 Calculating the Correlation

- To calculate the ***correlation r*** between two variables that are stored in vectors `x` and `y`, we use the function:

```
cor()       # Computes the correlation between two variables stored in
            # vectors x and y
```

- For example:

```
x <- c(60, 69, 66, 64, 54, 67, 59, 65, 63)
y <- c(136, 198, 194, 140, 93, 172, 116, 174, 145)
```

```
cor(x, y)

## [1] 0.9436756
```

Thus the ***correlation*** between `x` and `y` is $\boldsymbol{r = 0.9437}$.

1.2 t Test and Confidence Interval for a Population Correlation ρ

- To test whether an observed correlation is statistically significantly different from 0, i.e. to test

$$
\begin{aligned}
H_0 : \rho &= 0 \\
H_a : \rho &\neq 0
\end{aligned}
$$

where ρ is the true (unknown) population correlation, we use the function:

```
cor.test()      # Carries out a t test for the population correlation
                # using two variables stored in vectors x and y
```

- For example, using the vectors `x` and `y` from above:

```
cor.test(x, y)
```

```
cor.test(x, y)

##
##  Pearsons product-moment correlation
##
## data:  x and y
## t = 7.5459, df = 7, p-value = 0.0001321
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7489030 0.9883703
## sample estimates:
##       cor
## 0.9436756
```

From the output,

  - The observed ***t test*** statistic value is $\boldsymbol{t = 7.5459}$.
  - The ***p-value*** is $\mathbf{1.3 \times 10^{-4}}$ (from the t distribution with $\boldsymbol{n - 2 = 7}$ degrees of freedom).
  - The ***95% confidence interval*** for ρ is $\mathbf{(0.7489, 0.9884)}$.

2 Simple Linear Regression

2.1 Fitting the Model and Conducting t Tests for β_0 and β_1

- To fit the simple linear regression model

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{1}$$

and carry the associated t tests for the slope β_1 and intercept β_1, we use the "linear model" function:

```
lm()      # Fit a linear regression model and carry out a regression
          # analysis
```

To obtain a summary of the regression analysis, we use:

```
summary()      # Prints a summary of a linear regression analysis
               # carried out by lm()
```

`lm()` takes a *formula* as it's main argument and an optional argument `data` (a *data frame* containing the variables used in the *formula*).

- For example, to fit the model (1) to the data in the `x` and `y` *vectors*

```
x <- c(60, 69, 66, 64, 54, 67, 59, 65, 63)
y <- c(136, 198, 194, 140, 93, 172, 116, 174, 145)
```

we type:

```
my.reg <- lm(y ~ x)
```

Above, the *formula* `y ~ x` indicates that `y` is the response variable and `x` the predictor.

To see the results of the regression analysis we type:

```
summary(my.reg)

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.192  -7.233   2.849   5.727  20.424
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -301.0872    60.1885  -5.002 0.001561
## x             7.1919     0.9531   7.546 0.000132
##
## Residual standard error: 12.5 on 7 degrees of freedom
## Multiple R-squared:  0.8905,Adjusted R-squared:  0.8749
## F-statistic: 56.94 on 1 and 7 DF,  p-value: 0.0001321
```

From the output above, we conclude the following:

– The ***least squares estimates*** of the true (unknown) regression coefficients β_0 and β_1 are $\boldsymbol{b_0 = -301.0872}$ and $\boldsymbol{b_1 = 7.1919}$. Thus the equation of the least squares regression line is

$$\hat{y} = -301.0872 + 7.1919x.$$

– The ***standard errors*** of the estimates are

$$\begin{aligned} \boldsymbol{s\{b_0\}} &= \mathbf{60.1885} \\ \boldsymbol{s\{b_1\}} &= \mathbf{0.9531} \end{aligned}$$

– The test statistic for the ***t test*** of

$$\begin{aligned} H_0 : \beta_0 &= 0 \\ H_a : \beta_0 &\neq 0 \end{aligned}$$

is $\boldsymbol{t = b_0/s\{b_0\} = -5.002}$ and the p-value is **0.0016** (from the t distribution with $\boldsymbol{n-2=7}$ degrees of freedom). Thus we reject H_0 and conclude that β_0 is different from 0.

- The test statistic for the ***t test*** of

$$\begin{aligned} H_0 : \beta_1 &= 0 \\ H_a : \beta_1 &\neq 0 \end{aligned} \tag{2}$$

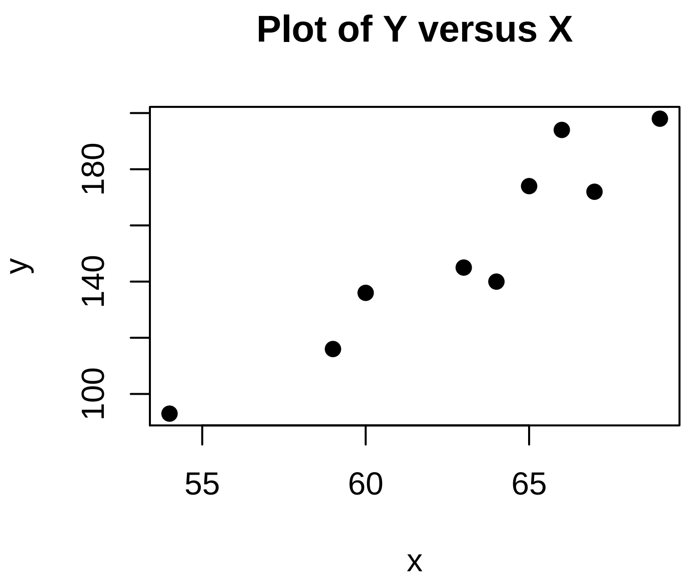
is $\boldsymbol{t = b_1/s\{b_1\} = 7.546}$, and from the t distribution with $\boldsymbol{n-2=7}$ df, the ***p-value*** is 1.3×10^{-4}. Thus we reject H_0 and conclude that β_1 is different from 0.

- The square root of the ***mean squared error*** is labeled `Residual standard error`, and its value is $\boldsymbol{\sqrt{\text{MSE}} = 12.5}$.
- The ***coefficient of determination*** is $\boldsymbol{R^2 = 0.8905}$, labeled `Multiple R-squared`.
- The test statistic for the ***regression model F test*** of the hypotheses (2) is $\boldsymbol{F = \text{MSR}/\text{MSE} = 56.9408}$. From the F distribution with numerator and denominator degrees of freedom **1** and **7**, respectively, the ***p-value*** is 1.3×10^{-4}. Thus we reject H_0 and conclude that β_1 is different from 0.

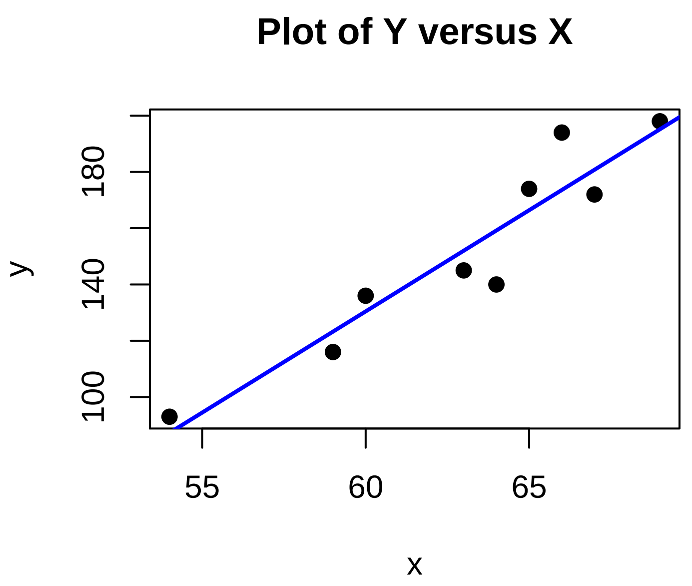
2.2 Plotting the Regression Line with the Data

- To plot the data in a scatterplot and add the regression line to the plot, we use `plot()` and `abline()` as follows:

```
plot(x, y, pch = 19, main = "Plot of Y versus X")
```



```
abline(my.reg, col = "blue", lwd = 2)
```



2.3 Computing Confidence Intervals for β_0 and β_1

- Once a regression model has been fit to the data using `lm()`, we can compute a $\mathbf{100(1-\alpha)\%}$ ***confidence interval for*** β_1

  $$b_1 \pm t_{\alpha/2}\, s\{b_1\}$$

  and also the $\mathbf{100(1-\alpha)\%}$ ***confidence interval for*** β_0

  $$b_0 \pm t_{\alpha/2}\, s\{b_0\}$$

  using the function:

  ```
  confint()       # Computes confidence intervals for regression model
                  # parameters (slope and intercept) from an lm object
  ```

  The `confint()` function takes an *lm* object (as returned by `lm()`) as its main argument, and an optional argument `level` specifying the level of confidence.

- For example, using the vectors `x` and `y` from above, to compute 95% confidence intervals for β_1 and β_0, we type:

```
my.reg <- lm(y ~ x)
```

```
confint(my.reg, level = 0.95)

##                   2.5 %      97.5 %
## (Intercept) -443.410309 -158.764110
## x              4.938183    9.445538
```

From the output:

- The ***95% confidence interval for*** β_0 is **(-443.4103, -158.7641)**.
- The ***95% confidence interval for*** β_1 is **(4.9382, 9.4455)**.

2.4 The Regression ANOVA Table

- To view the ***regression ANOVA table***, use the function:

```
anova()        # Prints a regression ANOVA table from the regression
               # analysis carried out by lm()
```

- For example, using the *lm* object `my.reg` from above:

```
anova(my.reg)

## Analysis of Variance Table
##
## Response: y
##           Df Sum Sq Mean Sq F value    Pr(>F)
## x          1 8896.3  8896.3  56.941 0.0001321
## Residuals  7 1093.7   156.2
```

From the output, we get:

- The ***regression sum of squares*** is **SSR = 8896.3** and the ***degrees of freedom*** for SSR is **1**.
- The ***error sum of squares*** is **SSE = 1093.7** and the ***degrees of freedom*** for SSE is **7**.
- The ***mean squares*** are **MSR = 8896.3** and **MSE = 156.2**.
- The test statistic for the ***regression model F test*** is $\boldsymbol{F}$ **= MSR/MSE = 56.941, *NA*** and the ***p-value*** (from the F distribution with numerator and denominator ***degrees of freedom*** **1** and **7**, respectively) is 1.3×10^{-4}**, NA**.

2.5 Obtaining the Residuals and Fitted Values

- Objects such as `my.reg` that belong to the *lm* class of objects are are really just *lists* that contain certain elements related to the regression analysis:

```
class(my.reg)

## [1] "lm"
```

```
is.list(my.reg)

## [1] TRUE
```

To see what elements `my.reg` contains, type:

```
names(my.reg)

##  [1] "coefficients"  "residuals"     "effects"
##  [4] "rank"          "fitted.values" "assign"
##  [7] "qr"            "df.residual"   "xlevels"
## [10] "call"          "terms"         "model"
```

As with any *list*, to extract specific named elements, we use the `$` operator. We could also use double square brackets `[[ ]]`. For example, to extract the ***estimated regression coefficients***, type:

```
my.reg$coefficients               # We could also type my.reg[["coefficients"]]

## (Intercept)           x
##  -301.08721     7.19186
```

and to get the ***residuals*** and ***fitted (predicted) values***, type:

```
my.reg$residuals

##          1          2          3          4          5
##   5.575581   2.848837  20.424419 -19.191860   5.726744
##          6          7          8          9
##  -8.767442  -7.232558   7.616279  -7.000000
```

and

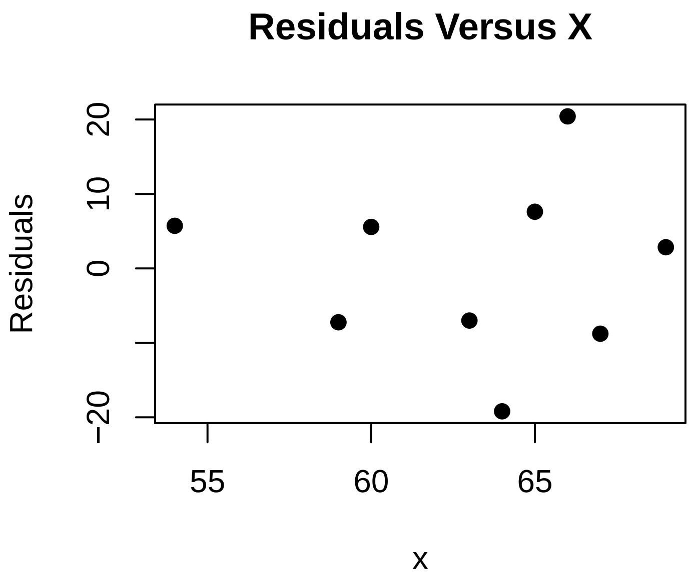
```
my.reg$fitted.values

##         1         2         3         4         5
## 130.42442 195.15116 173.57558 159.19186  87.27326
##         6         7         8         9
## 180.76744 123.23256 166.38372 152.00000
```

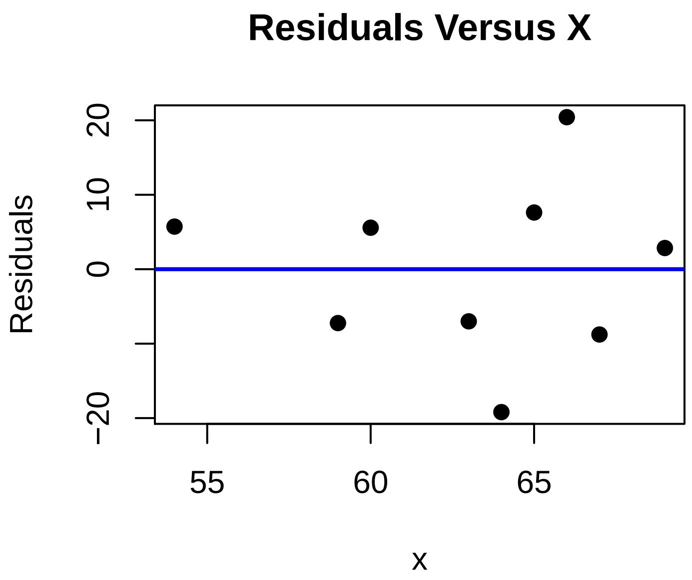
2.6 Checking Assumptions

- To plot the residuals versus the predictor (`x`) values and add a horizontal line at $y = 0$, type:

```
plot(x, my.reg$residuals, pch = 19, ylab = "Residuals",
    main = "Residuals Versus X")
```

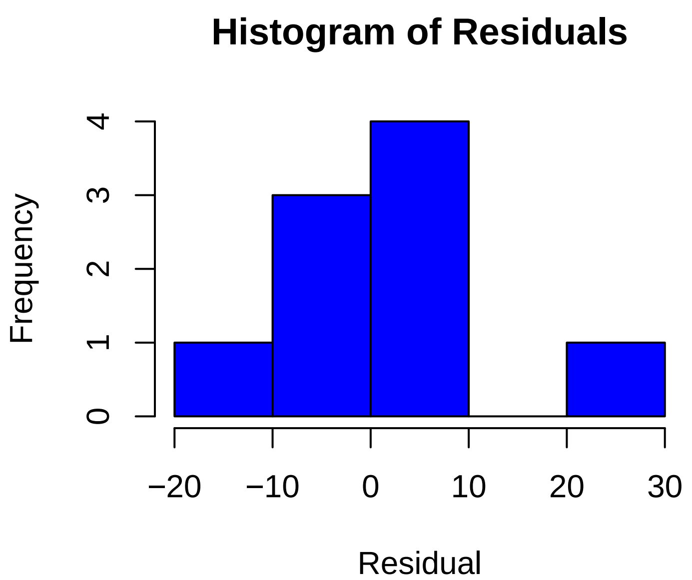



```
abline(h = 0, col = "blue", lwd = 2)
```



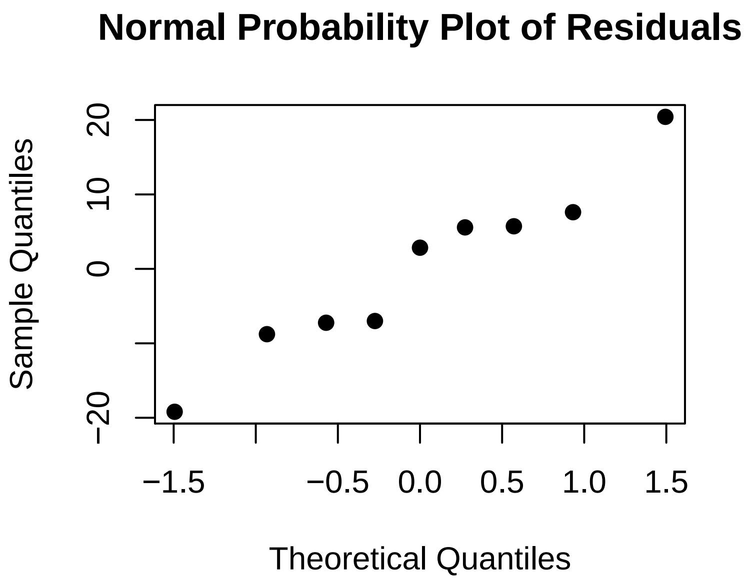
To make a histogram of the residuals, type:

```
hist(my.reg$residuals, xlab = "Residual",
    main = "Histogram of Residuals", col = "blue")
```

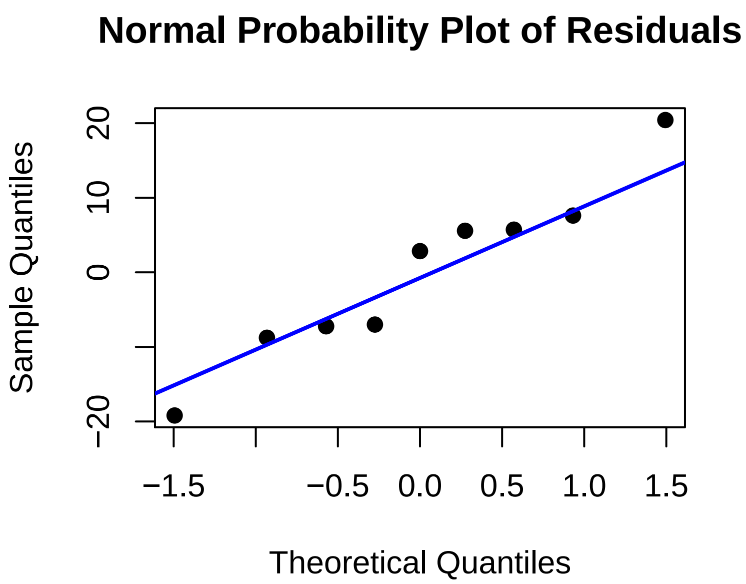


and to make a normal probability plot of the residuals and add a line to the plot, type:

```
qqnorm(my.reg$residuals, pch = 19,
    main = "Normal Probability Plot of Residuals")
```




```
qqline(my.reg$residuals, col = "blue", lwd = 2)
```



2.7 Predicting Y for a New Value of X

- To use a fitted regression line to ***predict*** Y at one or more *new* values of X (i.e. ones that aren't necessarily in the data set), we use:

```
predict()       # Uses a fitted regression model to predict Y for
                # one or more given values of X.  Can also be used to
                # compute a confidence interval for a mean response
                # and a prediction interval for a new Y.
```

  `predict()` takes as its main argument an *lm* object (such as `my.reg`) along with an argument `newdata`, a *data frame* containing the new X value(s).

- For example, to obtain the ***predicted value***

$$\hat{Y}_h = b_0 + b_1(70)$$

  at the new X value $X_h = 70$, we first create a data frame, `my.new.x` say, containing this new X value:

```
my.new.x <- data.frame(x = 70)
```

  The variable in this data frame **must have the same name** as the predictor variable that was used to create the *lm* object. For example, because the object `my.reg` was created using the call `lm(y ~ x)`, the variable in `my.new.x` must be named `x`.

  Now we pass this newly created *data frame* to `predict()` along with `my.reg` to get the predicted value

```
predict(my.reg, newdata = my.new.x)

##       1
## 202.343
```

  Thus the ***predicted value*** is $\boldsymbol{\hat{Y}_h = -301.087 + 7.192(70) = 202.343}$.

- Predictions at more than one new X value are obtained in a similar manner by including those X values in the *data frame* `my.new.x`, for example:

```
my.new.x <- data.frame(x = c(70, 75))
```

```
predict(my.reg, newdata = my.new.x)

##        1        2
## 202.3430 238.3023
```

2.8 Confidence Interval for a Mean Response

- A **100(1 − α)%** ***confidence interval for the mean response*** $E(Y_h) = \beta_0 + \beta_1 X_h$, at a given value X_h of the predictor variable X, is also created using `predict()`, but now we specify `interval = "confidence"` for the optional argument `interval`.
- For example, to create a 95% confidence interval for the mean response when the value of the predictor is $X_h = 70$, we first create a *data frame* containing this new X value:

```
my.new.x <- data.frame(x = 70)
```

then pass it to `predict()` along with our *lm* object `my.reg`:

```
predict(my.reg, newdata = my.new.x, interval = "confidence", level = 0.95)

##       fit      lwr      upr
## 1 202.343 183.7435 220.9425
```

The ***fitted value*** **202.343** serves as an estimate of the mean response $E(Y_h)$. The ***confidence interval*** (endpoints `lwr` and `upr`) for $E(Y_h)$ is **(183.744, 220.943)**, and we can be 95% confident that $E(Y_h)$ is in this interval.

2.9 Prediction Interval for a New *Y*

- To compute a **100(1 − α)%** ***prediction interval*** for a new response Y_h at the value $X = X_h$, we use `prdict()` as above, but specify `interval = "prediction"`.
- For example, below we obtain a 95% prediction interval for Y_h at the new X value $X_h = 70$:

```
predict(my.reg, newdata = my.new.x, interval = "prediction", level = 0.95)

##       fit      lwr      upr
## 1 202.343 167.4211 237.2649
```

In this case, the ***fitted value*** **202.343** serves as a prediction for the value of Y_h. The ***95% prediction interval*** (endpoints `lwr`, `upr`) for Y_h is **(167.421, 237.265)**, and we can be 95% confident that the new Y_h value will fall in this interval.