

Statistics

Overview

We will cover how to use R to compute some of basic statistics and fit some basic statistical models.

- Correlation
- T-test
- Linear Regression
- Logistic Regression

DISCLAIMER: We will focus on how to use R software to do these. We will be glossing over the statistical theory and “formulas” for these tests. Moreover, we do not claim the data we use for demonstration meet assumptions of the methods.

There are plenty of resources online for learning more about these methods, as well as dedicated Biostatistics series (at different advancement levels) at the JHU School of Public Health.

Correlation

Correlation

Function `cor()` computes correlation in R

```
cor(x, y = NULL, use = "everything",  
    method = c("pearson", "kendall", "spearman"))
```

To use:

- provide two numeric vectors (arguments `x`, `y`) to compute correlation between them, or
- provide matrix or data frame (argument `x`) that has at least 2 columns (must be numeric) to compute correlation between all pairs

By default, Pearson correlation coefficient is computed.

Correlation

https://jhudatascience.org/intro_to_r/data/Charm_City_Circulator_Ridership.csv

```
circ <- jhur::read_circulator()
head(circ)
```

```
# A tibble: 6 × 15
```

	day	date	orangeBoardings	orangeAlightings	orangeAverage	purpleBoardings
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	Monday	01/1...	877	1027	952	NA
2	Tuesday	01/1...	777	815	796	NA
3	Wednesday	01/1...	1203	1220	1212.	NA
4	Thursday	01/1...	1194	1233	1214.	NA
5	Friday	01/1...	1645	1643	1644	NA
6	Saturday	01/1...	1457	1524	1490.	NA

```
# ... with 9 more variables: purpleAlightings <dbl>, purpleAverage <dbl>,  
#   greenBoardings <dbl>, greenAlightings <dbl>, greenAverage <dbl>,  
#   bannerBoardings <dbl>, bannerAlightings <dbl>, bannerAverage <dbl>,  
#   daily <dbl>
```

Correlation for two vectors

First, we compute correlation by providing two vectors.

Like other functions, if there are **NAs**, you get **NA** as the result. But if you specify use only the complete observations, then it will give you correlation using the non-missing data.

```
x <- pull(circ, orangeAverage)
y <- pull(circ, purpleAverage)
```

```
cor(x, y)
```

```
[1] NA
```

```
cor(x, y, use = "complete.obs")
```

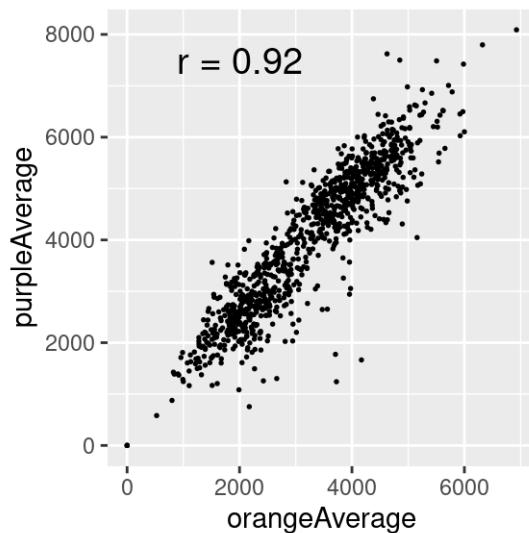
```
[1] 0.9195356
```

Correlation for two vectors with plot

Note that you can add the correlation value to a plot, via the `annotate()`.

```
cor_val <- cor(x, y, use = "complete.obs")  
cor_val_label <- paste0("r = ", round(cor_val, 3))
```

```
circ %>%  
  ggplot(aes(x = orangeAverage, y = purpleAverage)) +  
  geom_point(size = 0.3) +  
  annotate("text", x = 2000, y = 7500, label = cor_val_label, size = 5)
```



Correlation for data frame columns

We can compute correlation for all pairs of columns of a data frame / matrix. We typically just say, *"compute correlation matrix"*.

Columns must be all numeric!

```
circ_subset_Average <- circ %>% select(ends_with("Average"))  
dim(circ_subset_Average)
```

```
[1] 1146    4
```

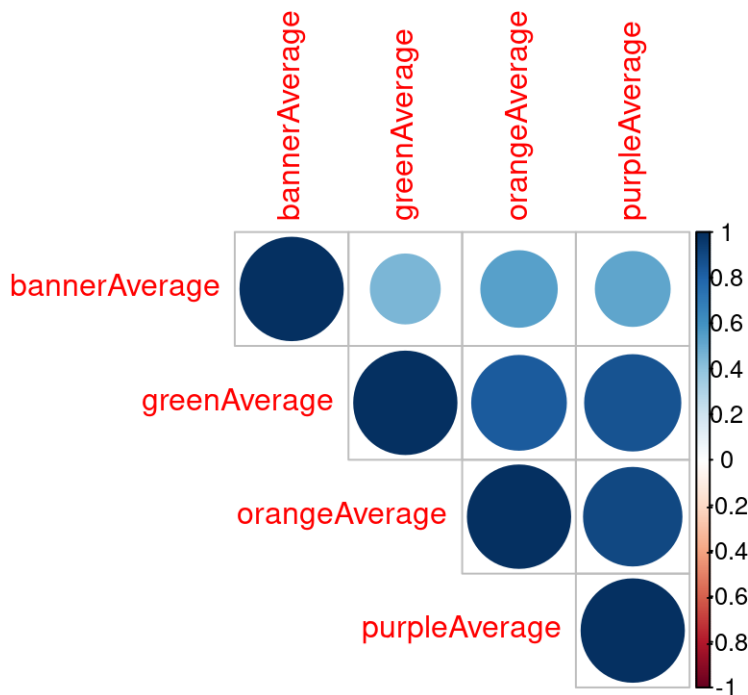
```
cor_mat <- cor(circ_subset_Average, use = "complete.obs")  
cor_mat
```

	orangeAverage	purpleAverage	greenAverage	bannerAverage
orangeAverage	1.0000000	0.9078826	0.8395806	0.5447031
purpleAverage	0.9078826	1.0000000	0.8665630	0.5213462
greenAverage	0.8395806	0.8665630	1.0000000	0.4533421
bannerAverage	0.5447031	0.5213462	0.4533421	1.0000000

Correlation for data frame columns with plot

- Google, *"r correlation matrix plot"*

```
library(corrplot)
corrplot(cor_mat, type = "upper", order = "hclust")
```



Lab Part 1

[Website](#)

T-test

T-test

The commonly used are:

- **one-sample t-test** – used to test mean of a variable in one group
- **two-sample t-test** – used to test difference in means of a variable between two groups (if the “two groups” are data of the *same* individuals collected at 2 time points, we say it is two-sample paired t-test)

The `t.test()` function in R is one to address the above.

```
t.test(x, y = NULL,  
       alternative = c("two.sided", "less", "greater"),  
       mu = 0, paired = FALSE, var.equal = FALSE,  
       conf.level = 0.95, ...)
```

Running one-sample t-test

It tests mean of a variable in one group. By default (i.e., without us explicitly specifying values of other arguments):

- tests whether a mean of a variable is equal to 0 ($\mu=0$)
- uses “two sided” alternative (`alternative = "two.sided"`)
- returns result assuming confidence level 0.95 (`conf.level = 0.95`)

```
x <- pull(circ, orangeAverage)
t.test(x)
```

One Sample t-test

```
data: x
t = 83.279, df = 1135, p-value < 0.000000000000000022
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 2961.700 3104.622
sample estimates:
mean of x
 3033.161
```

Running two-sample t-test

It tests test difference in means of a variable between two groups. By default:

- tests whether difference in means of a variable is equal to 0 ($\mu=0$)
- uses “two sided” alternative (`alternative = "two.sided"`)
- returns result assuming confidence level 0.95 (`conf.level = 0.95`)
- assumes data are not paired (`paired = FALSE`)
- assumes true variance in the two groups is not equal (`var.equal = FALSE`)

```
x <- pull(circ, orangeAverage)
y <- pull(circ, purpleAverage)
t.test(x, y)
```

Welch Two Sample t-test

```
data: x and y
t = -17.076, df = 1984, p-value < 0.000000000000000022
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1096.7602 -870.7867
sample estimates:
```

T-test: retrieving information from the result

Object returned from `t.test()` function is a named list. We can use it to access test result elements. The easiest way to do this is to use base R (\$) notation).

```
result <- t.test(x, y)
is.list(result)
```

```
[1] TRUE
```

```
names(result)
```

```
[1] "statistic"    "parameter"    "p.value"      "conf.int"     "estimate"
[6] "null.value"   "stderr"       "alternative"   "method"       "data.name"
```

```
result$statistic
```

-17.07579 t

```
result$p.value
```

[illegible]

T-test: retrieving information from the result with **broom** package

The **broom** package has a `tidy()` function that can organize results into a data frame so that they are easily manipulated (or nicely printed)

```
library(broom)

result <- t.test(x, y)
result_tidy <- tidy(result)
result_tidy

# A tibble: 1 × 10
  estimate estimate1 estimate2 statistic p.value parameter conf.low conf.high
  <dbl>      <dbl>      <dbl>      <dbl>   <dbl>      <dbl>      <dbl>      <dbl>
1   -984.      3033.      4017.     -17.1 4.20e-61    1984.     -1097.     -871.
# ... with 2 more variables: method <chr>, alternative <chr>
```


Some other statistical tests

- `wilcox.test()` – Wilcoxon signed rank test, Wilcoxon rank sum test
- `shapiro.test()` – Shapiro test
- `ks.test()` – Kolmogorov-Smirnov test
- `var.test()` – Fisher's F-Test
- `chisq.test()` – Chi-squared test

Lab Part 2

[Website](#)

Regression

Linear regression

Linear regression is a method to model the relationship between a response and one or more explanatory variables.

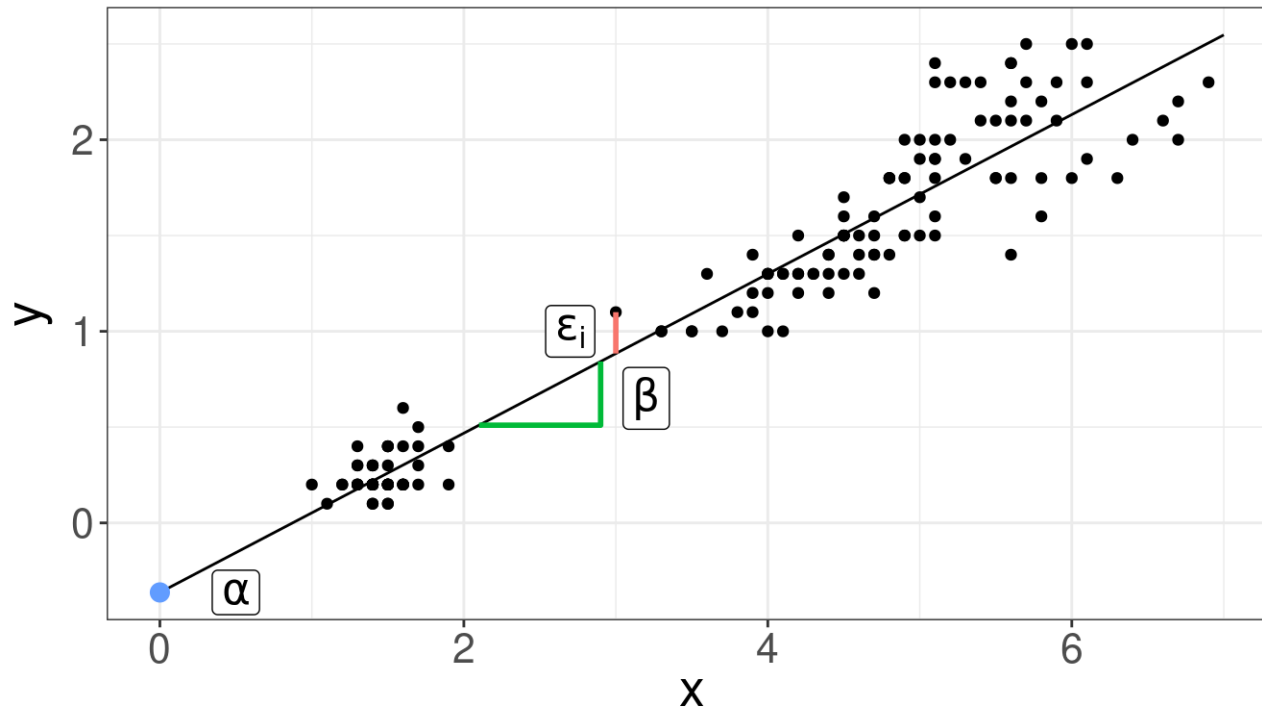
We provide a little notation here so some of the commands are easier to put in the proper context.

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

where:

- y_i is the outcome for person i
- α is the intercept
- β is the slope
- x_i is the predictor for person i
- ε_i is the residual variation for person i

Linear regression



Linear regression

Linear regression is a method to model the relationship between a response and one or more explanatory variables.

We provide a little notation here so some of the commands are easier to put in the proper context.

$$y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \varepsilon_i$$

where:

- y_i is the outcome for person i
- α is the intercept
- $\beta_1, \beta_2, \beta_3$ are the slopes for variables x_{i1}, x_{i2}, x_{i3}
- x_{i1}, x_{i2}, x_{i3} are the predictors for person i
- ε_i is the residual variation for person i

Linear regression fit in R

To fit linear models in R, we use function `lm()`.

```
lm(formula, data, subset, weights, na.action,  
    method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,  
    singular.ok = TRUE, contrasts = NULL, offset, ...)
```

We typically provide two arguments:

- `formula` – model formula written using names of columns in our data
- `data` – our data frame

Linear regression fit in R: model formula

Model formula

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

translates to `y ~ x` in R formula for this example.

In practice, `y` and `x` are replaced with the **names of columns from our data set**.

- For example, if we want to fit a regression model where outcome is `income` and predictor is `years_of_education`, our formula would be:

```
income ~ years_of_education
```


Linear regression fit in R: model formula

Model formula

$$y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \varepsilon_i$$

translates to `y ~ x1 + x2 + x3` in R formula for this example.

In practice, `y` and `x1`, `x2`, `x3` are replaced with the **names of columns from our data set**.

- For example, if we want to fit a regression model where outcome is `income` and predictors are `years_of_education`, `age`, `location` then our formula would be:

```
income ~ years_of_education + age + location
```

Linear regression

We will use `kaggleCarAuction.csv` dataset from one of the Kaggle competitions.

https://jhudatascience.org/intro_to_r/data/kaggleCarAuction.csv

```
cars <- jhur::read_kaggle()
head(cars)
```

```
# A tibble: 6 × 34
```

	RefId	IsBadBuy	PurchDate	Auction	VehYear	VehicleAge	Make	Model	Trim	SubModel
	<dbl>	<dbl>	<chr>	<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>	<chr>
1	1	0	12/7/2009	ADESA	2006	3	MAZDA	MAZD...	i	4D SEDA...
2	2	0	12/7/2009	ADESA	2004	5	DODGE	1500...	ST	QUAD CA...
3	3	0	12/7/2009	ADESA	2005	4	DODGE	STRA...	SXT	4D SEDA...
4	4	0	12/7/2009	ADESA	2004	5	DODGE	NEON	SXT	4D SEDAN
5	5	0	12/7/2009	ADESA	2005	4	FORD	FOCUS	ZX3	2D COUP...
6	6	0	12/7/2009	ADESA	2004	5	MIT...	GALA...	ES	4D SEDA...

```
# ... with 24 more variables: Color <chr>, Transmission <chr>, WheelTypeID <chr>,
# WheelType <chr>, VehOdo <dbl>, Nationality <chr>, Size <chr>,
# TopThreeAmericanName <chr>, MMRAcquisitionAuctionAveragePrice <chr>,
# MMRAcquisitionAuctionCleanPrice <chr>,
# MMRAcquisitionRetailAveragePrice <chr>,
```

Linear regression: model fitting

We fit linear regression model with vehicles odometer (distance traveled by a vehicle; `VehOdo`) as an outcome and vehicle (`VehicleAge`) age as a predictor.

```
fit <- lm(VehOdo ~ VehicleAge, data = cars)
print(fit)
```

Call:

```
lm(formula = VehOdo ~ VehicleAge, data = cars)
```

Coefficients:

(Intercept)	VehicleAge
60127	2723

Linear regression: model summary

The `summary()` command returns a list that shows us some more detail

```
sfit <- summary(fit)
print(sfit)
```

Call:

```
lm(formula = VehOdo ~ VehicleAge, data = cars)
```

Residuals:

Min	1Q	Median	3Q	Max
-71097	-9500	1383	10323	41037

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	60127.24	134.80	446.04	<0.00000000000000002 ***
VehicleAge	2722.94	29.86	91.18	<0.00000000000000002 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13810 on 72981 degrees of freedom

Multiple R-squared: 0.1023, Adjusted R-squared: 0.1023

F-statistic: 8314 on 1 and 72981 DF, p-value: < 0.000000000000000022

Linear regression: retrieving information with **broom** package

Use `tidy` to create a tibble with the coefficient estimates.

tidy() is a function from the broom package

```
tidy(sfit)
```

A tibble: 2 × 5

	term	estimate	std.error	statistic	p.value
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	(Intercept)	60127.	135.	446.	0
2	VehicleAge	2723.	29.9	91.2	0

Linear regression: model summary

Model summary is a named list and we can access its specific elements. Again, we should use base R (\$) notation).

```
names(sfit)
```

```
[1] "call"          "terms"          "residuals"      "coefficients"  
[5] "aliased"        "sigma"          "df"             "r.squared"  
[9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```
sfit$r.squared
```

```
[1] 0.1022682
```

Linear regression: multiple predictors

Let's try adding another explanatory variable to our model, Warranty price (WarrantyCost)

```
fit_2 <- lm(VehOdo ~ VehicleAge + WarrantyCost, data = cars)
summary(fit_2)
```

```
Call:
lm(formula = VehOdo ~ VehicleAge + WarrantyCost, data = cars)
```

Residuals:

Min	1Q	Median	3Q	Max
-67895	-8673	940	9305	45765

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	52422.92491	145.98775	359.09	<0.0000000000000002	***
VehicleAge	1944.65509	28.85619	67.39	<0.0000000000000002	***
WarrantyCost	8.58147	0.08251	104.01	<0.0000000000000002	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12890 on 72980 degrees of freedom

Multiple R-squared: 0.2182, Adjusted R-squared: 0.2181

F-statistic: 1.018e+04 on 2 and 72980 DF, p-value: < 0.00000000000000022

Linear regression: factors

Factors get special treatment in regression models - lowest level of the factor is the comparison group, and all other factors are relative to its values.

`TopThreeAmericanName` states if the manufacturer is one of the top three American manufacturers.

```
top_3 <- pull(cars, TopThreeAmericanName)
table(top_3)
```

```
top_3
CHRYSLER    FORD      GM      NULL    OTHER
   23399    12315    25314         5    11950
```


Linear regression: factors

```
fit_3 <- lm(VehOdo ~ factor(TopThreeAmericanName), data = cars)
summary(fit_3)
```

```
Call:
lm(formula = VehOdo ~ factor(TopThreeAmericanName), data = cars)
```

Residuals:

Min	1Q	Median	3Q	Max
-71947	-9634	1532	10472	45936

Coefficients:

	Estimate	Std. Error	t value
(Intercept)	68248.48	92.98	733.984
factor(TopThreeAmericanName)FORD	8523.49	158.35	53.828
factor(TopThreeAmericanName)GM	4952.18	128.99	38.393
factor(TopThreeAmericanName)NULL	-2004.68	6361.60	-0.315
factor(TopThreeAmericanName)OTHER	584.87	159.92	3.657

	Pr(> t)
(Intercept)	< 0.00000000000000002 ***
factor(TopThreeAmericanName)FORD	< 0.00000000000000002 ***
factor(TopThreeAmericanName)GM	< 0.00000000000000002 ***
factor(TopThreeAmericanName)NULL	0.752670
factor(TopThreeAmericanName)OTHER	0.000255 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14220 on 72978 degrees of freedom

Multiple R-squared: 0.04822, Adjusted R-squared: 0.04817

F-statistic: 924.3 on 4 and 72978 DF, p-value: < 0.000000000000000022

Linear regression: retrieving information with **broom** package

```
tidy(fit_3)
```

```
# A tibble: 5 × 5
```

term	estimate	std.error	statistic	p.value
<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1 (Intercept)	68248.	93.0	734.	0
2 factor(TopThreeAmericanName)FORD	8523.	158.	53.8	0
3 factor(TopThreeAmericanName)GM	4952.	129.	38.4	2.74e-319
4 factor(TopThreeAmericanName)NULL	-2005.	6362.	-0.315	7.53e- 1
5 factor(TopThreeAmericanName)OTHER	585.	160.	3.66	2.55e- 4

Generalized Linear Models (GLMs)

Generalized Linear Models (GLMs) allow for fitting regressions for non-continuous/normal outcomes. Examples include: logistic regression, Poisson regression.

We fit GLM with a `glm()` function that has a very similar syntax to the `lm()` function.

The primary difference is in `glm()`, we additionally specify the `family` argument – a description of the error distribution and link function to be used in the model. These include:

- `binomial(link = "logit")`
- `poisson(link = "log")`, and other.

See `?family` documentation for details of family functions.

Logistic regression

IsBadBuy is a 0/1-valued variable stating “if the kicked vehicle was an avoidable purchase”.

```
glm_fit <- glm(IsBadBuy ~ VehOdo + VehicleAge, data = cars, family = binomial())
summary(glm_fit)
```

```
Call:
glm(formula = IsBadBuy ~ VehOdo + VehicleAge, family = binomial(),
    data = cars)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-0.9943	-0.5481	-0.4534	-0.3783	2.6318

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)	
(Intercept)	-3.7782285193	0.0638091954	-59.211	<0.00000000000000002	***
VehOdo	0.0000083410	0.0000008526	9.783	<0.00000000000000002	***
VehicleAge	0.2681085873	0.0067722363	39.589	<0.00000000000000002	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 54421 on 72982 degrees of freedom
Residual deviance: 52346 on 72980 degrees of freedom
AIC: 52352

Number of Fisher Scoring iterations: 5

Final note

Some final notes:

- Researcher's responsibility to **understand the statistical method** they use – underlying assumptions, correct interpretation of method results
- Researcher's responsibility to **understand the R software** they use – meaning of function's arguments and meaning of function's output elements

Lab Part 3

[Website](#)