Statistics

Module 11

Andrew Jaffe

June 18, 2015

Statistics

Now we are going to cover how to perform a variety of basic statistical tests in R.

- Correlation
- T-tests
- Linear Regression
- Logistic Regression
- Proportion tests
- Chi-squared
- Fisher's Exact Test

Note: We will be glossing over the statistical theory and "formulas" for these tests. There are plenty of resources online for learning more about these tests, as well as dedicated Biostatistics series at the School of Public Health

Correlation

`cor()` performs correlation in R

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

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 on the non-missing data.

```
> load("../data/charmcirc.rda")
> cor(circ2$orangeAverage, circ2$purpleAverage)
```

```
[1] NA
```

```
> cor(circ2$orangeAverage, circ2$purpleAverage, use="complete.obs")
```

```
[1] 0.9195356
```

Correlation

You can also get the correlation between matrix columns

```
> signif(cor(circ2[,grep("Average",names(circ2))], use="complete.obs"),3)
```

```
              orangeAverage purpleAverage greenAverage bannerAverage
orangeAverage         1.000         0.908        0.840         0.545
purpleAverage         0.908         1.000        0.867         0.521
greenAverage          0.840         0.867        1.000         0.453
bannerAverage         0.545         0.521        0.453         1.000
```

Or between columns of two matrices, column by column.

```
> signif(cor(circ2[,3:4],circ2[,5:6], use="complete.obs"),3)
```

```
              greenAverage bannerAverage
orangeAverage        0.840         0.545
purpleAverage        0.867         0.521
```

Correlation

You can also use cor.test() to test for whether correlation is significant (ie non-zero). Note that linear regression may be better, especially if you want to regress out other confounders.

```
> ct= cor.test(circ2$orangeAverage,
+      circ2$purpleAverage, use="complete.obs")
> ct
```
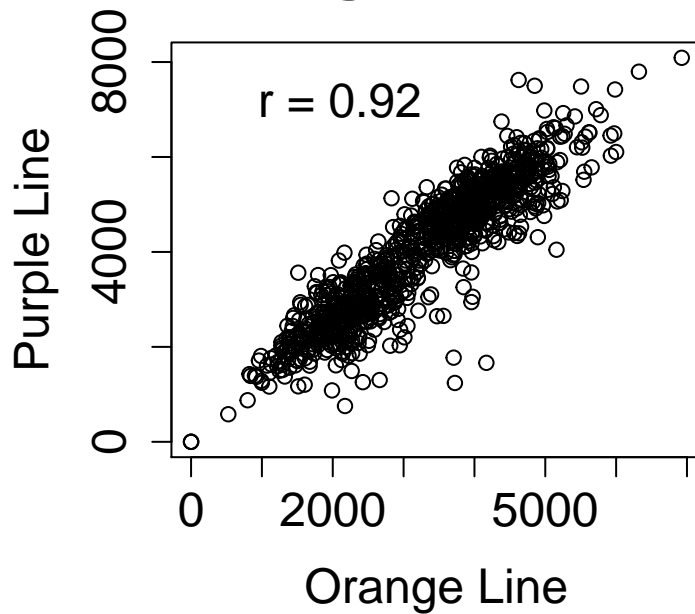
```
	Pearson's product-moment correlation

data:  circ2$orangeAverage and circ2$purpleAverage
t = 73.656, df = 991, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9093438 0.9286245
sample estimates:
      cor
0.9195356
```

Correlation

Note that you can add the correlation to a plot, via the legend() function.

```
> plot(circ2$orangeAverage, circ2$purpleAverage,
+      xlab="Orange Line", ylab="Purple Line",
+      main="Average Ridership",cex.axis=1.5,
+      cex.lab=1.5,cex.main=2)
> legend("topleft", paste("r =", signif(ct$estimate,3)), bty="n",cex=1.5)
```

Average Ridership

Correlation

For many of these testing result objects, you can extract specific slots/results as numbers, as the `ct` object is just a list.

```
> # str(ct)
> names(ct)
```

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

```
> ct$statistic
```

```
       t
73.65553
```

```
> ct$p.value
```

```
[1] 0
```

T-tests

The T-test is performed using the `t.test()` function, which essentially tests for the difference in means of a variable between two groups.

In this syntax, x and y are the column of data for each group.

```
> tt = t.test(circ2$orangeAverage, circ2$purpleAverage)
> tt
```

```
    Welch Two Sample t-test

data:  circ2$orangeAverage and circ2$purpleAverage
t = -17.076, df = 1984, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1096.7602  -870.7867
sample estimates:
mean of x mean of y
 3033.161  4016.935
```

T-tests

t.test saves a lot of information: the difference in means estimate, confidence interval for the difference conf.int, the p-value p.value, etc.

```
> names(tt)
```

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

T-tests

You can also use the 'formula' notation. In this syntax, it is y ~ x, where x is a factor with 2 levels or a binary variable and y is a vector of the same length.
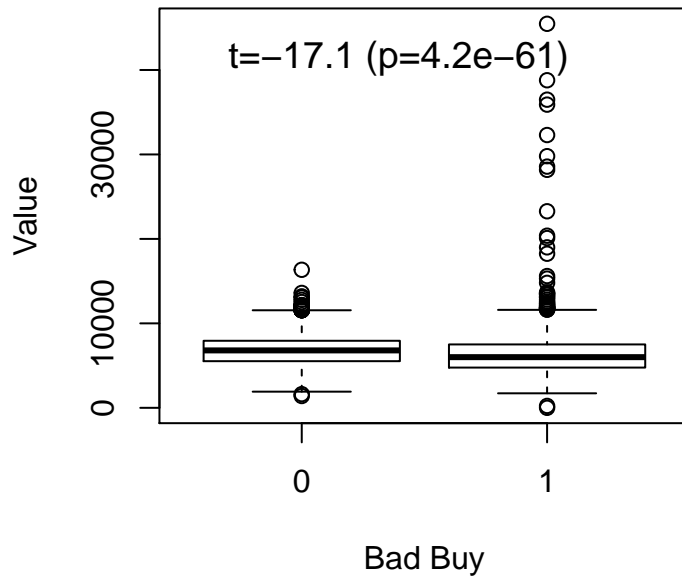
```
> cars = read.csv("../data/kaggleCarAuction.csv",as.is=TRUE)
> tt2 = t.test(VehBCost~IsBadBuy, data=cars)
> tt2$estimate
```

```
mean in group 0 mean in group 1
      6797.077        6259.274
```

T-tests

You can add the t-statistic and p-value to a boxplot.

```
> boxplot(VehBCost~IsBadBuy, data=cars,
+         xlab="Bad Buy",ylab="Value")
> leg = paste("t=", signif(tt$statistic,3),
+     " (p=",signif(tt$p.value,3),")",sep="")
> legend("topleft", leg, cex=1.2, bty="n")
```

Linear Regression

Now we will briefly cover linear regression. I will use 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

The `R` version of the regression model is:

```
y ~ x
```

where:

- y is your outcome
- x is/are your predictor(s)

Linear Regression

For a linear regression, when the predictor is binary this is the same as a t-test:

```
> fit = lm(VehBCost~IsBadBuy, data=cars)
> fit
```

```
Call:
lm(formula = VehBCost ~ IsBadBuy, data = cars)

Coefficients:
(Intercept)      IsBadBuy
     6797.1        -537.8
```

'(Intercept)' is α

'IsBadBuy' is β

Linear Regression

The `summary` command gets all the additional information (p-values, t-statistics, r-square) that you usually want from a regression.

```
> sfit = summary(fit)
> print(sfit)
```

```
Call:
lm(formula = VehBCost ~ IsBadBuy, data = cars)

Residuals:
   Min     1Q Median     3Q    Max
 -6258  -1297    -27   1153  39210

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 6797.077      6.953  977.61   <2e-16 ***
IsBadBuy    -537.803     19.826  -27.13   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1759 on 72981 degrees of freedom
Multiple R-squared:  0.009982,  Adjusted R-squared:  0.009969
F-statistic: 735.9 on 1 and 72981 DF,  p-value: < 2.2e-16
```

Linear Regression

The coefficients from a `summary` are the coefficients, standard errors, t-statistcs, and p-values for all the estimates.

```
> names(sfit)


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

> sfit$coef


            Estimate Std. Error    t value     Pr(>|t|)
(Intercept) 6797.0774   6.952728 977.61299  0.00000e+00
IsBadBuy    -537.8033  19.825525 -27.12681 3.01661e-161
```

Linear Regression
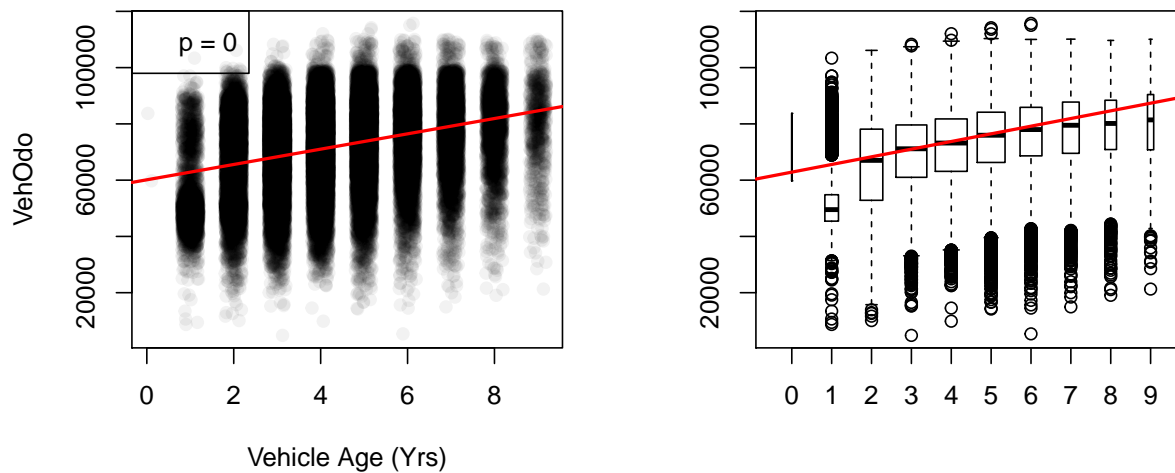
We'll look at vehicle odometer value by vehicle age:

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


##
## Call:
## lm(formula = VehOdo ~ VehicleAge, data = cars)
##
## Coefficients:
## (Intercept)    VehicleAge
##       60127          2723
```

Linear Regression

We can visualize the vehicle age/odometer relationshp using scatter plots or box plots (with regression lines). The function `abline` will plot the regresion line on the plot.

```
> library(scales) # we need this for the alpha command - make points transparent
> par(mfrow=c(1,2))
> plot(VehOdo ~ jitter(VehicleAge,amount=0.2), data=cars, pch = 19,
+      col = alpha("black",0.05), xlab="Vehicle Age (Yrs)")
> abline(fit, col="red",lwd=2)
> legend("topleft", paste("p =",summary(fit)$coef[2,4]))
> boxplot(VehOdo ~ VehicleAge, data=cars, varwidth=TRUE)
> abline(fit, col="red",lwd=2)
```

Linear Regression

Note that you can have more than 1 predictor in regression models. The interpretation for each slope is change in the predictor corresponding to a one-unit change in the outcome, holding all other predictors constant.

```
> fit2 = lm(VehOdo ~ IsBadBuy + VehicleAge, data=cars)
> summary(fit2)
```

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

Residuals:
   Min     1Q Median     3Q    Max
-70856  -9490   1390  10311  41193

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 60141.77     134.75  446.33   <2e-16 ***
IsBadBuy     1329.00     157.84    8.42   <2e-16 ***
VehicleAge   2680.33      30.27   88.53   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13810 on 72980 degrees of freedom
Multiple R-squared:  0.1031,    Adjusted R-squared:  0.1031
F-statistic:  4196 on 2 and 72980 DF,  p-value: < 2.2e-16
```
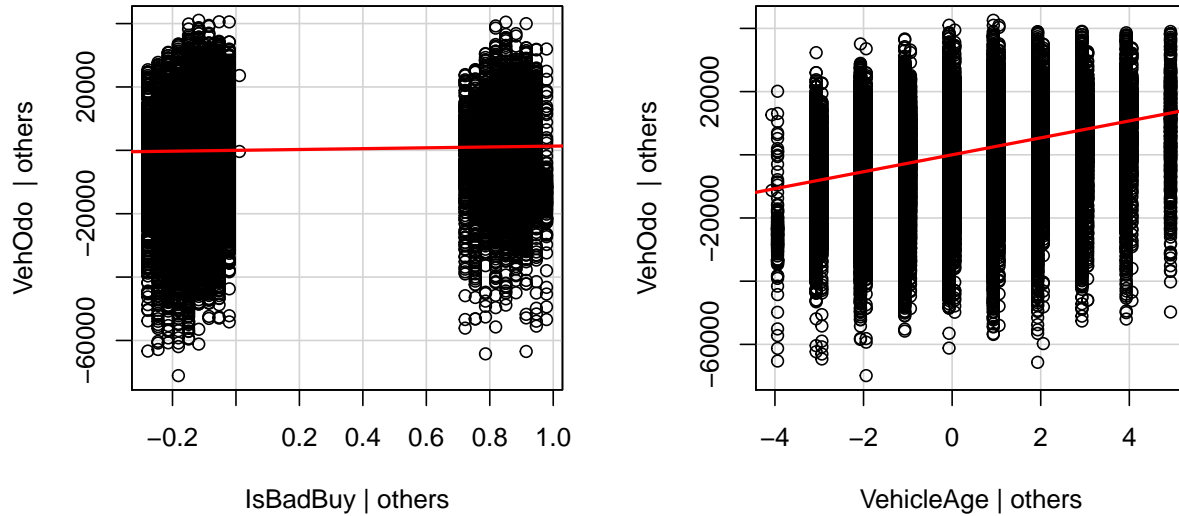
Linear Regression

Added-Variable plots can show you the relationship between a variable and outcome after adjusting for other variables. The function `avPlots` from the `car` package can do this:
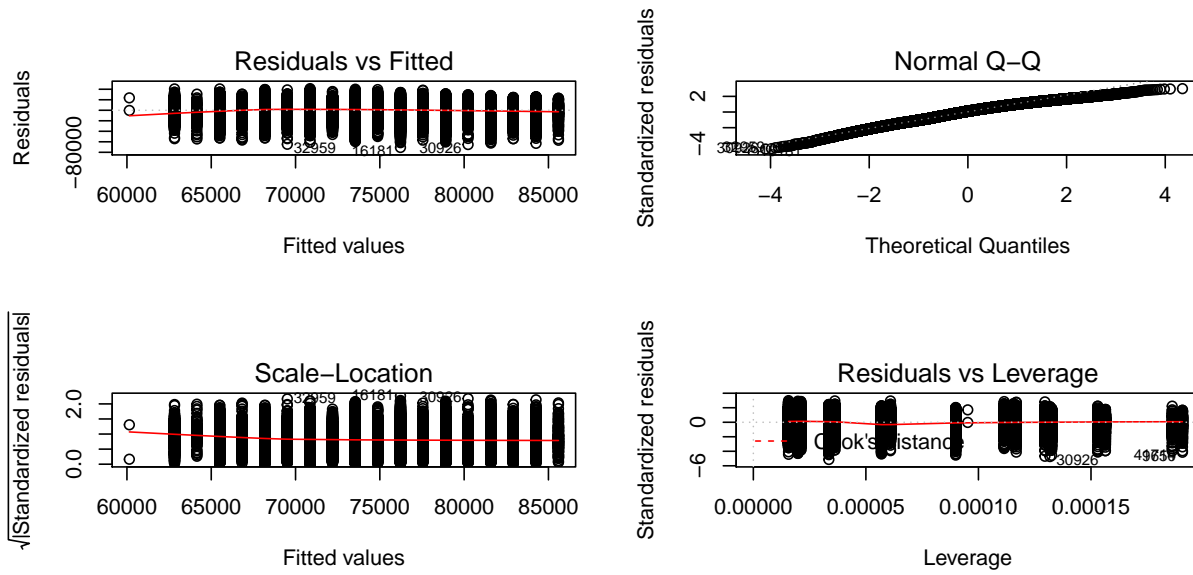
```
> library(car)
> avPlots(fit2)
```

Added−Variable Plots

Linear Regression

Plot on an `lm` object will do diagnostic plots. Residuals vs. Fitted should have no discernable shape (the red line is the smoother), the qqplot shows how well the residuals fit a normal distribution, and Cook's distance measures the influence of individual points.

```
> par(mfrow=c(2,2))
> plot(fit2, ask= FALSE)
```

Linear Regression

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.

```
> fit3 = lm(VehOdo ~ factor(TopThreeAmericanName), data=cars)
> summary(fit3)
```

```
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 Pr(>|t|)
(Intercept)                     68248.48      92.98 733.984  < 2e-16 ***
factor(TopThreeAmericanName)FORD 8523.49     158.35  53.828  < 2e-16 ***
factor(TopThreeAmericanName)GM   4952.18     128.99  38.393  < 2e-16 ***
factor(TopThreeAmericanName)NULL -2004.68   6361.60  -0.315 0.752670
factor(TopThreeAmericanName)OTHER 584.87     159.92   3.657 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: < 2.2e-16
```

Logistic Regression and GLMs

Generalized Linear Models (GLMs) allow for fitting regressions for non-continous/normal outcomes. The `glm` has similar syntax to the `lm` command. Logistic regression is one example.

```
> glmfit = glm(IsBadBuy ~ VehOdo + VehicleAge, data=cars, family=binomial())
> summary(glmfit)
```

```
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) -3.778e+00  6.381e-02 -59.211   <2e-16 ***
VehOdo       8.341e-06  8.526e-07   9.783   <2e-16 ***
VehicleAge   2.681e-01  6.772e-03  39.589   <2e-16 ***
---
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
```

Logistic Regression

Note the coefficients are on the original scale, we must exponentiate them for odds ratios:

```
> exp(coef(glmfit))
```

```
(Intercept)      VehOdo  VehicleAge
 0.02286316  1.00000834  1.30748911
```

Proportion tests

`prop.test()` can be used for testing the null that the proportions (probabilities of success) in several groups are the same, or that they equal certain given values.

```
prop.test(x, n, p = NULL,
          alternative = c("two.sided", "less", "greater"),
          conf.level = 0.95, correct = TRUE)
```

```
> prop.test(x=15, n =32)
```

```
	1-sample proportions test with continuity correction

data:  15 out of 32, null probability 0.5
X-squared = 0.03125, df = 1, p-value = 0.8597
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.2951014 0.6496695
sample estimates:
      p
0.46875
```

Chi-squared tests

chisq.test() performs chi-squared contingency table tests and goodness-of-fit tests.

```
chisq.test(x, y = NULL, correct = TRUE,
           p = rep(1/length(x), length(x)), rescale.p = FALSE,
           simulate.p.value = FALSE, B = 2000)
```

```
> tab = table(cars$IsBadBuy, cars$IsOnlineSale)
> tab
```

```
        0     1
  0 62375  1632
  1  8763   213
```

Chi-squared tests

You can also pass in a table object (such as tab here)

```
> cq=chisq.test(tab)
> cq
```

```
	Pearson's Chi-squared test with Yates' continuity correction

data:  tab
X-squared = 0.92735, df = 1, p-value = 0.3356
```

```
> names(cq)
```

```
[1] "statistic" "parameter" "p.value"   "method"    "data.name" "observed"
[7] "expected"  "residuals" "stdres"
```

```
> cq$p.value
```

```
[1] 0.3355516
```

Chi-squared tests

Note that does the same test as `prop.test`, for a 2x2 table.

```
> chisq.test(tab)
```

```
    Pearson's Chi-squared test with Yates' continuity correction

data:  tab
X-squared = 0.92735, df = 1, p-value = 0.3356
```

```
> prop.test(tab)
```

```
    2-sample test for equality of proportions with continuity
    correction

data:  tab
X-squared = 0.92735, df = 1, p-value = 0.3356
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.005208049  0.001673519
sample estimates:
   prop 1     prop 2
0.9745028 0.9762701
```

Fisher's Exact test

`fisher.test()` performs contingency table test using the hypogeometric distribution (used for small sample sizes).

```
fisher.test(x, y = NULL, workspace = 200000, hybrid = FALSE,
            control = list(), or = 1, alternative = "two.sided",
            conf.int = TRUE, conf.level = 0.95,
            simulate.p.value = FALSE, B = 2000)
```

```
> fisher.test(tab)
```

```
    Fisher's Exact Test for Count Data

data:  tab
p-value = 0.3324
alternative hypothesis: true odds ratio is not equal to 1
```

```
95 percent confidence interval:
 0.8001727 1.0742114
sample estimates:
odds ratio
 0.9289923
```

Probability Distributions

Sometimes you want to generate data from a distribution (such as normal), or want to see where a value falls in a known distribution. R has these distibutions built in:

- Normal
- Binomial
- Beta
- Exponential
- Gamma
- Hypergeometric
- etc

Probability Distributions

Each has 4 options:

- r for random number generation [e.g. rnorm()]
- d for density [e.g. dnorm()]
- p for probability [e.g. pnorm()]
- q for quantile [e.g. qnorm()]

```
> rnorm(5)
```

```
[1]  0.2138222  0.4995846  0.8617045 -1.2882207  1.5599225
```

Sampling

The sample() function is pretty useful for permutations

```
> sample(1:10, 5, replace=FALSE)
```

```
[1] 6 2 4 3 8
```

Also, if you want to only plot a subset of the data (for speed/time or overplotting)

```
> samp.cars <- cars[ sample(nrow(cars), 10000), ]
> plot(VehOdo ~ jitter(VehBCost,amount=0.3), data= samp.cars)
```