COMPUTATIONAL FINANCE & RISK MANAGEMENT
UNIVERSITY of WASHINGTON
Department of Applied Mathematics

Linear Models in R
The `lm(.)` function

CFRM 425 (006)

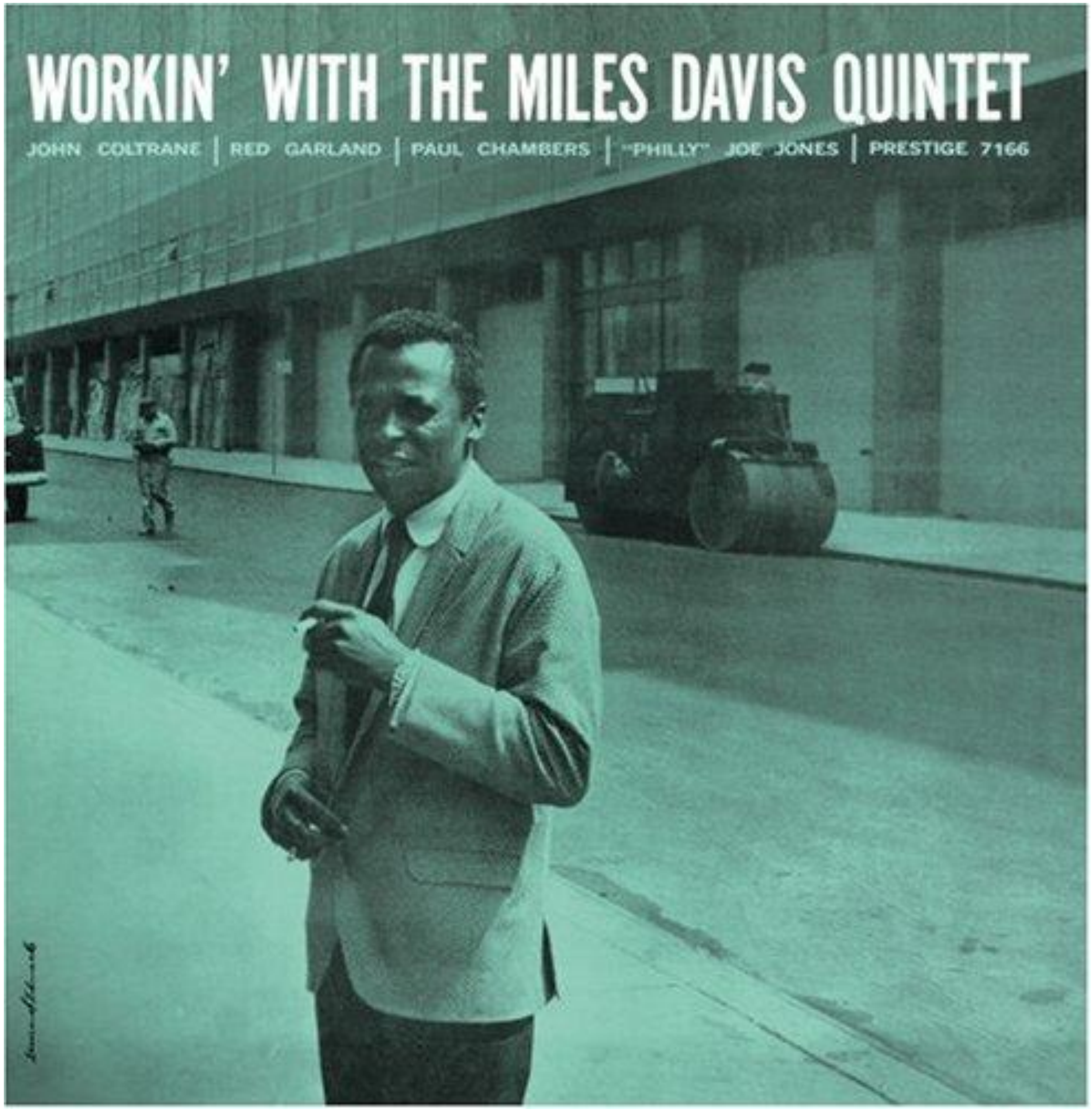
R Programming for Quantitative Finance

References/Reading/Topics

- [JV], sections on simple and multiple regression in Ch's 2 & 3
- These slides
- Kabacoff, https://www.statmethods.net/stats/regression.html

- Topics:
  - Working with the **`lm(.)`** function
  - Plotting regression results

Working with the `lm(.)` function



The `lm(.)` Function

- For simple or multiple linear regression
- Assumes the usual least squares and normality conditions
- Returns an `lm` object (`list` type) containing member variables (not exhaustive):
  - The least squares coefficients
  - Residuals
  - Degrees of freedom for the residuals
  - Fitted values of the response (y) for each predictor value (x)

```
"coefficients"  "residuals"      "effects"
"rank"          "fitted.values"  "assign"
"qr"            "df.residual"    "xlevels"
"call"          "terms"          "model"
```

- These may be access directly from the object with the $ operator, or in functions provided in R; eg, if `fit` is an `lm` object, then the following are equivalent:
  - **`fit$coefficients`** **`coefficients(fit)`**
  - **`fit$residuals`** **`residuals(fit)`**
  - **`fit$df.residual`** **`df.residual(fit)`**
  - **`fit$fitted.values`** **`fitted.values(fit)`**

The `lm(.)` Function

- In addition, there are convenience functions that take in an `lm` object argument:
  - **`summary(fit)`**
  - **`anova(fit)`**
  - **`AIC(fit)`** (Akaike Information Criterion)
  - **`fitted(fit)`** (same as `fitted.values(fit)`)
- The **summary(.)** function returns a summary.lm object (also a list type), which holds the following member variables:

```
"call"           "terms"           "residuals"
"coefficients"   "aliased"         "sigma"
"df"             "r.squared"       "adj.r.squared"
"fstatistic"     "cov.unscaled"
```

- Note that there is some overlap with the member variables on the lm object
- In the case of coefficients, however, the summary provides results of t-tests on each regression variable, in addition to the coefficient values
- We will look at some specific examples in the sample code
- We will also look at the other three functions listed above
- First, let's look at a simple linear regression example (see sample code)

Simple Linear Regression

```
Call:
lm(formula = mpg ~ hp, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-5.7121 -2.1122 -0.8854  1.5819  8.2360 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
hp          -0.06823    0.01012  -6.742 1.79e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.863 on 30 degrees of freedom
Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892 
F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

Multiple Regression

- Now, let's extend the model and regress mpg on cyl, disp, hp, and wt
- Our code now becomes

```
mfit <- lm(mpg ~ cyl + disp + hp + wt, data = mtcars)

mfit$coefficients

coefficients(mfit)
anova(mfit)
vcov(mfit)
AIC(mfit)

msumm <- summary(mfit)
msumm$coefficients
```

- See sample code for results
- Again, note that there is some overlap (eg, three ways to retrieve the fitted regression coefficients)

Multiple Regression on All Available Predictors

- In this case, there is a convenient shortcut:

```
mfit2 <- lm(mpg ~ ., data = mtcars)
mfit2$coefficients
anova(mfit2)
```

- This works out OK as all predictor variables are numeric (integer values coerced to floating point)
- See results using sample code

Categorical Variables

- In *mtcars*, the gear column is integer valued and treated as a continuous numerical value in the regression
- It can have value 3, 4, or 5
- We can make it a categorical variable by using the **`as.factor(.)`** function inside the regression function; viz,

```
mfitCat2 <- lm(mpg ~ as.factor(gear) + cyl + disp + hp + wt - 1, data = mtcars)
```

- Note, however, that in the coefficient estimates are expressed as factors 2 and 3 (gears 4 and 5) relative to factor 1 (gear = 3):

```
coefficients(mfitCat)
    (Intercept) as.factor(gear)4 as.factor(gear)5             disp               hp               wt
   35.024485218      1.886537475      2.389772782      0.008576519     -0.040322982     -3.754332711
```

Categorical Variables

- A trick to remedy this is as follows:
  - Remove the degree of freedom used by the intercept term
  - This taken by the "left out" factor, which results in coefficient estimates for each possible categorical level
- This is done by placing a "-1" following the regression expression in the **lm(.)** function argument:

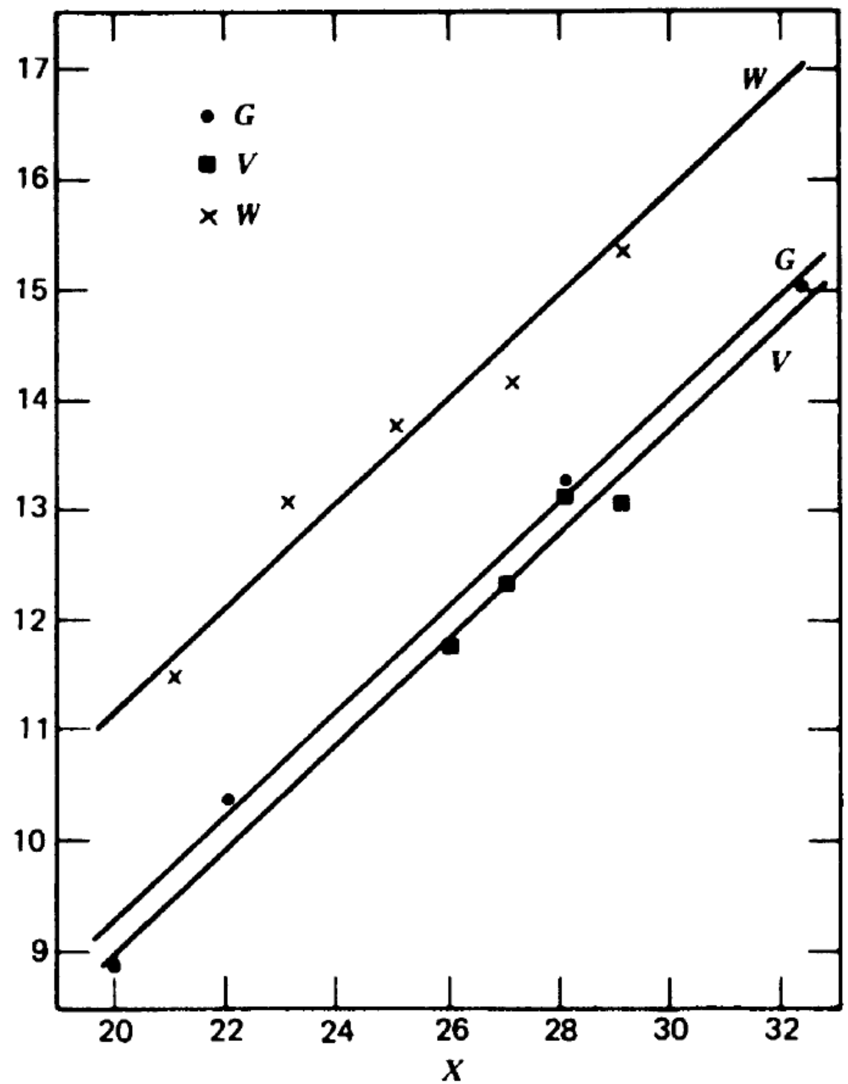
```
mfitCat2 <- lm(mpg ~ as.factor(gear) + cyl + disp + hp + wt - 1, data = mtcars)
```

- The coefficients are now calculated as follows:

```
coefficients(mfitCat2)
as.factor(gear)3 as.factor(gear)4 as.factor(gear)5              cyl             disp               hp               wt
     39.07251405      40.38188478      40.24039280      -1.12410697       0.01609356      -0.02572368      -3.92874891
```

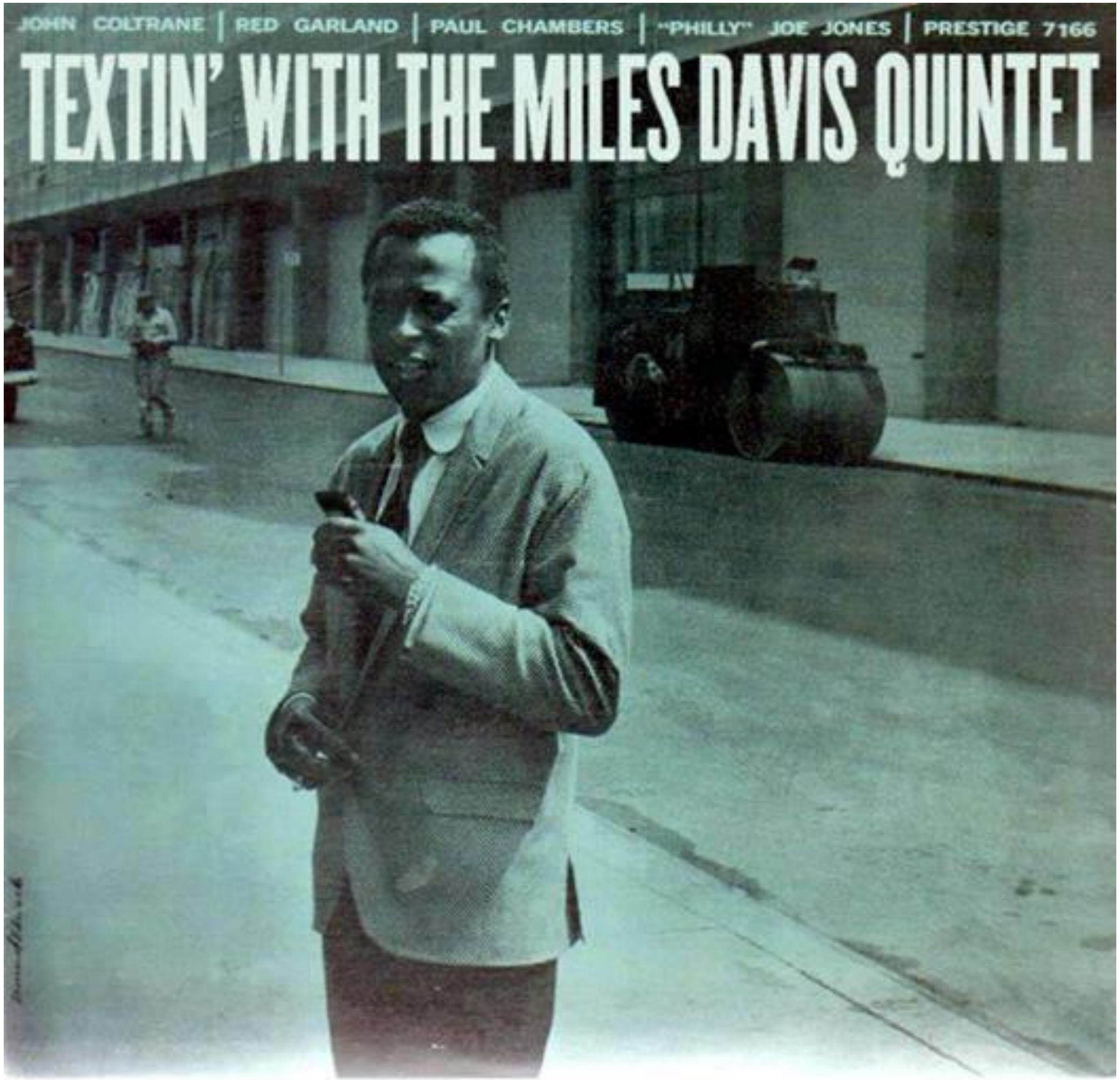
Categorical Variables

- Example: Multiple levels of a categorical variable in a linear regression



Draper & Smith, *Applied Regression Analysis (3E)*, Wiley, 1998, p 304

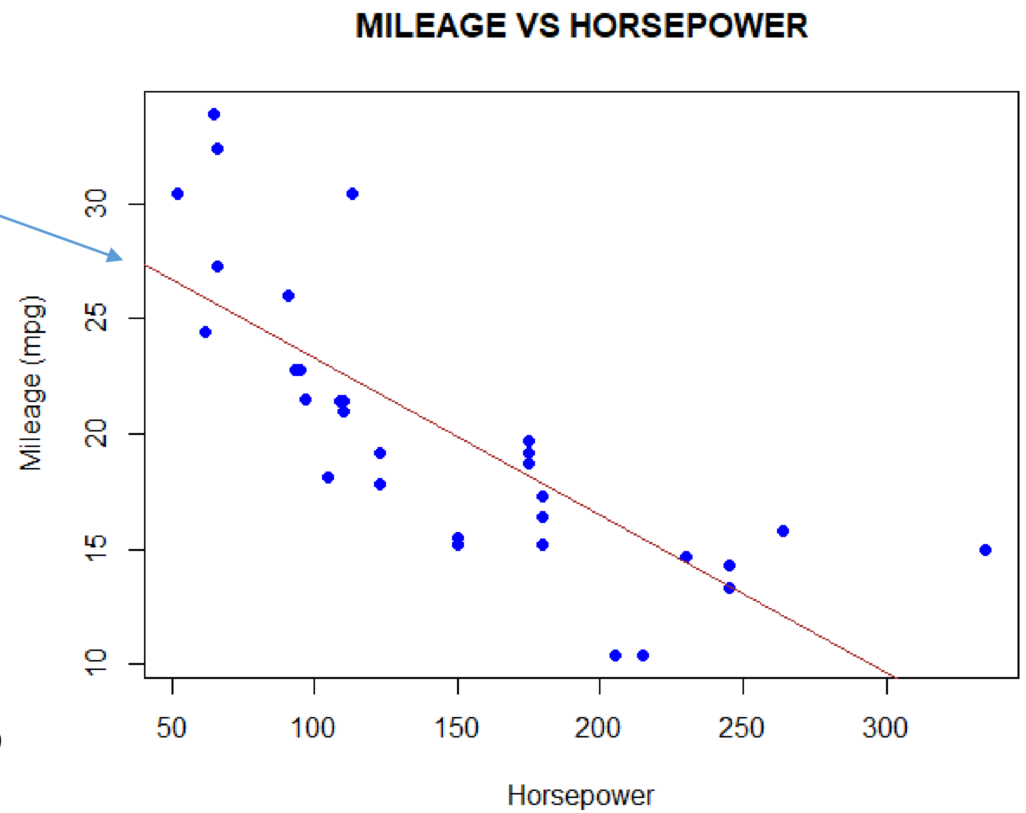
Plotting Regression Results



Plotting Simple Linear Regression

- Start with the simple 2-D case:

```
plot(x = hp, y = mpg, pch = 16, cex = 1.3, col = "blue",
     main = "MILEAGE VS HORSEPOWER", xlab = "Horsepower",
     ylab = "Mileage (mpg)")
# Overlay regression line; just put in lm object 'fit' as argument:
abline(reg = fit,
    col = "brown")
```



- **`abline(.)`**: *line from a to b*

Plotting Simple Linear Regression

- Now, look at predicted values of the response (y) for given predictor values (x)
- Choose a vector of x values, and calculate the estimated y value using the resulting regression equation
- Include *extrapolated values* below the minimum and above the maximum data values for x (just to make it more interesting);

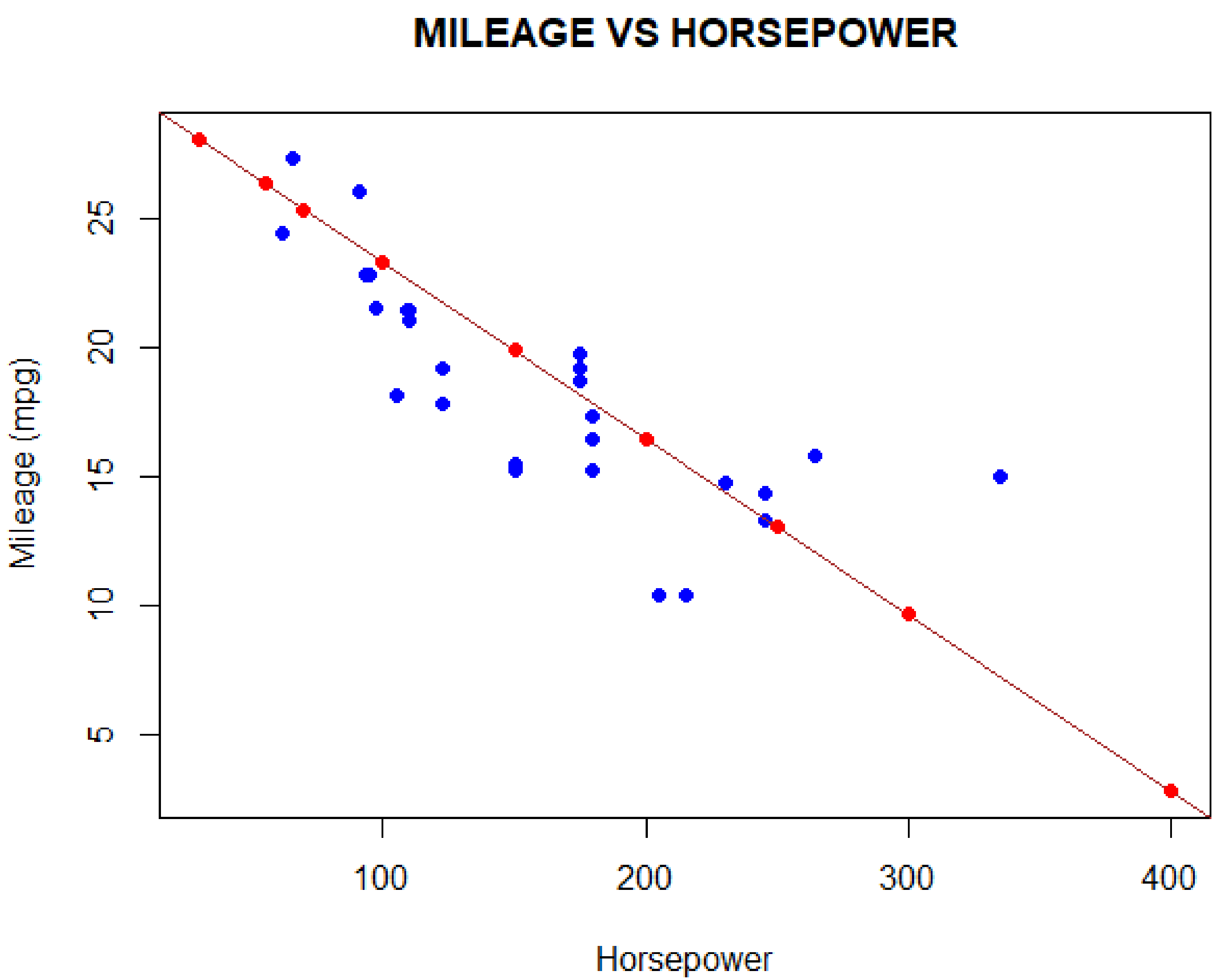
```
# Prediction:
min(hp)   # 52
max(hp)   # 335


# Note that the endpoints are extrapolating:
hpPred <- data.frame(hp=c(30, 55, 70, 100, 150, 200, 250, 300, 400))
pred <- predict(object = fit, newdata = hpPred)
```

Plotting Simple Linear Regression

- Overlay the predicted values on the original plot
- Note that we need to define the ranges for x and y, as we chose x values outside of the original data range:

```
plot(x = hp, y = mpg, pch = 16, cex = 1.0, col = "blue",
     main = "MILEAGE VS HORSEPOWER", xlab = "Horsepower",
     ylab = "Mileage (mpg)", xlim = c(min(hpPred$hp), max(hpPred$hp)),
     ylim = c(min(pred), max(pred)))
# Overlay regression line; reg is argument for a *regression object*:
abline(reg = fit, col = "brown")
# Use the lines(.) function to overlay the predicted values:
lines(y = pred, x = hpPred$hp, type = "p",
      pch = 16, cex = 1.3, col = "red")
```

- Result:

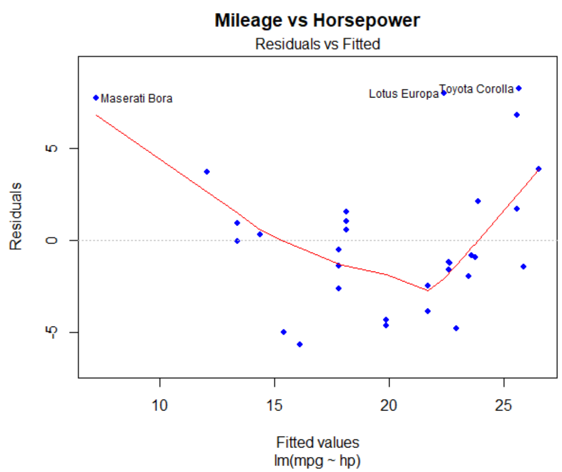

MILEAGE VS HORSEPOWER
Mileage (mpg)
5
10
15
20
25
100
200
300
400
Horsepower

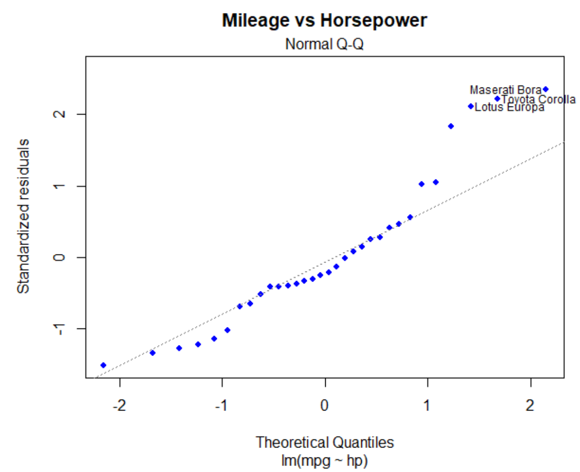
Plotting Simple Linear Regression

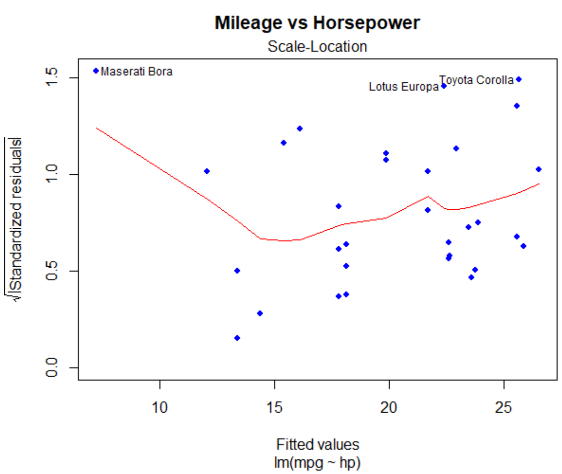
- We can also get various plots of the residuals by putting the lm object as the argument in plot (see **`plot.lm(.)`** overload in Help):

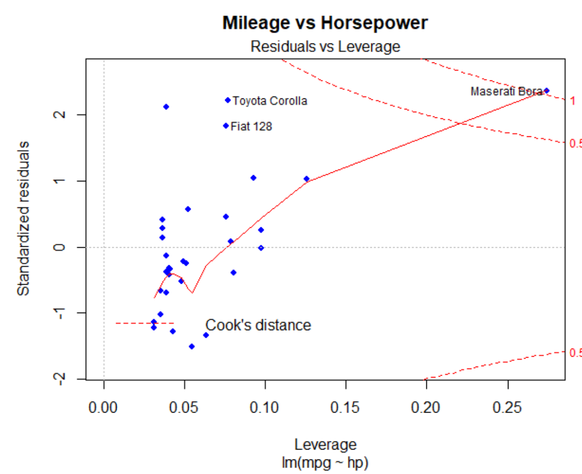
```
plot(x = fit, main = "Mileage vs Horsepower", pch = 16, cex = 0.8, col = "blue")
```

- To advance to the next plot, press Return in the R console (see code example)





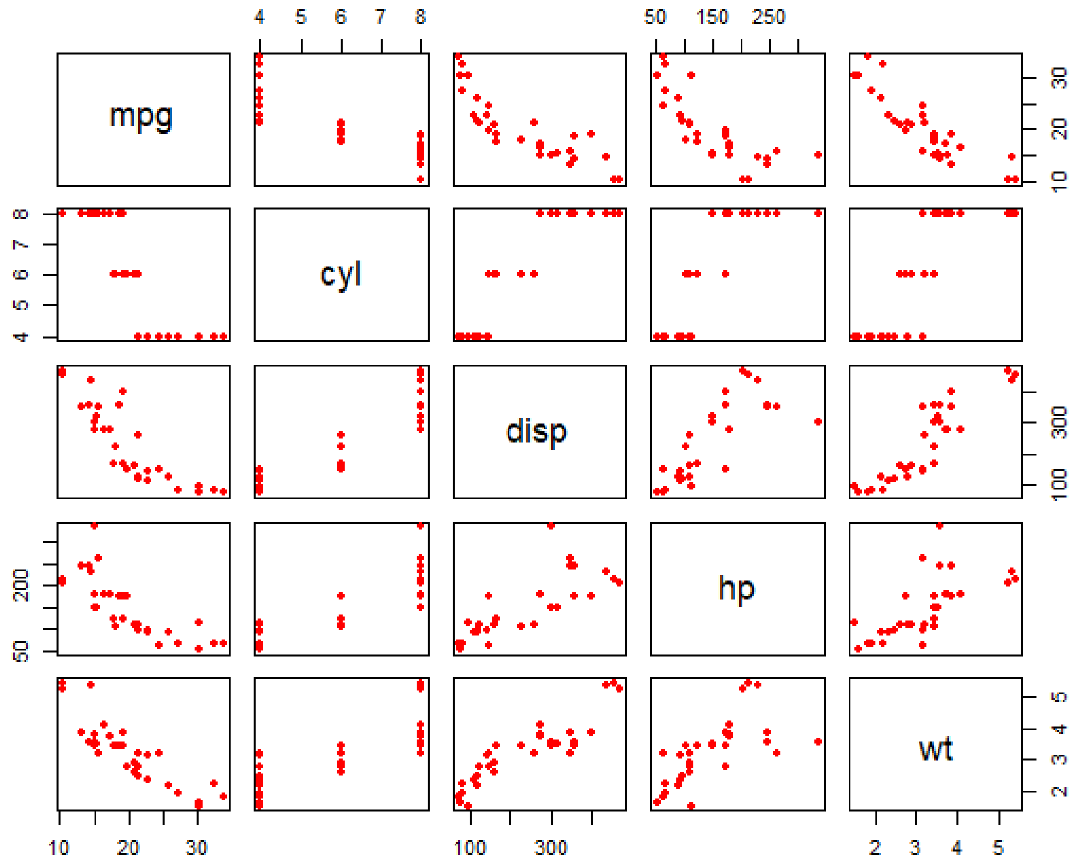




Plotting Multiple Linear Regression

- Multiple regression: Pairwise plots
- In Base R, use the pairs(.) function:

```
mfit <- lm(mpg ~ cyl + disp + hp + wt, data = mtcars)
pairs(cbind(mpg, cyl, disp, hp, wt), pch = 16, cex = 0.8, col = "red")
```



- There are of course many more ways to slice and dice multiple regression results into plots
- We will hold this topic in abeyance for now

[END]