Supervised learning - Regression 1

David Vichansky

30-11-2020

Here is an example file you can write.

First, load the packages:

```
library(ISLR)
library(MASS)
library(tidyverse)
```

```
## -- Attaching packages ------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ---------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::select() masks MASS::select()
```

```
library(haven)
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(boot)
```

```
##
## Attaching package: 'boot'

## The following object is masked from 'package:lattice':
```

```
##
##     melanoma
```

1. Create a linear model object called lm_ses using the formula medv ~ lstat and the Boston dataset.

```
Boston <- Boston

medv <- Boston$medv
lstat <- Boston$lstat

formula <- Boston$medv ~ lstat

lm.fit = lm(medv lstat)
lm.fit
```

```
##
## Call:
## lm(formula = medv ~ lstat)
##
## Coefficients:
## (Intercept)        lstat
##       34.55        -0.95
```

```
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.168  -3.990  -1.318   2.034  24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384    0.56263   61.41   <2e-16 ***
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

2. Use the function coef() to extract the intercept and slope from the lm_ses object. Interpret the slope coefficient.

```r
#names(lm.fit )

coef(lm.fit)
```

```
## (Intercept)       lstat 
##  34.5538409  -0.9500494
```

We observe a slight negative relation between 'medv' (the median price of the owners home) with lstat ('percent of low income homes in the area').

3. Use summary() to get a summary of the lm_ses object. What do you see? You can use the help file ?summary.lm.

```r
summary <- summary(formula)
summary
```

```
##  Length   Class    Mode 
##       3 formula    call
```

```r
class(formula)
```

```
## [1] "formula"
```

Seems to show a formula with 3 variables inside of it. We assume this is the interpect, the slope gradient and some error term.

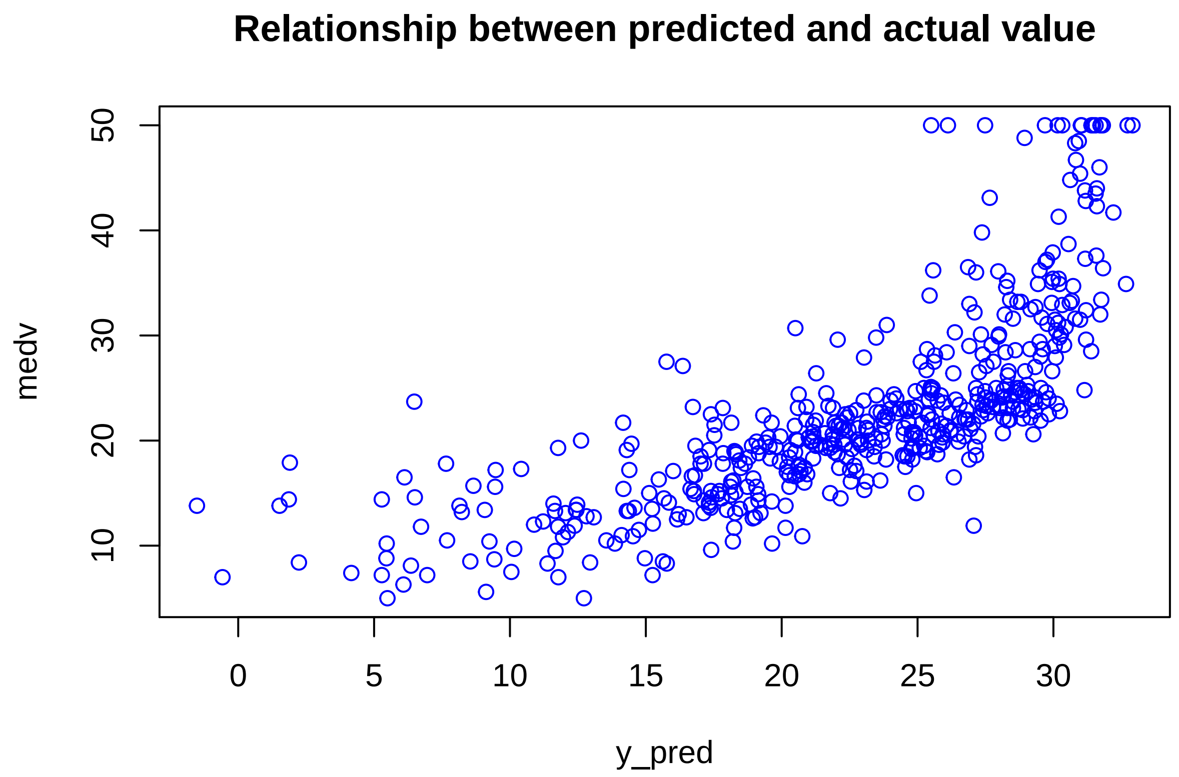
4. Save the predicted y values to a variable called y_pred.

```r
lstat <- Boston$lstat

y_pred = predict (lm.fit, data.frame(lstat))
```

5.Create a scatter plot with y_pred mapped to the x position and the true y value (Boston$medv) mapped to the y value. What do you see? What would this plot look like if the fit were perfect?

```r
plot(y_pred ,medv, type = "p",
main="Relationship between predicted and actual value",col="blue")
```



We observe that the predict value is good at estimating the median cost of a house at the lower end, but is not so good at doing so for when the actual median price of a neighbourhood is high!

6. Use the seq() function to generate a sequence of 1000 equally spaced values from 0 to 40. Store this vector in a data frame with (data.frame() or tibble()) as its column name lstat. Name the data frame pred_dat.

```
lstat <- c(seq(0,40,length=1000))
pred_dat <- data.frame(lstat)
```

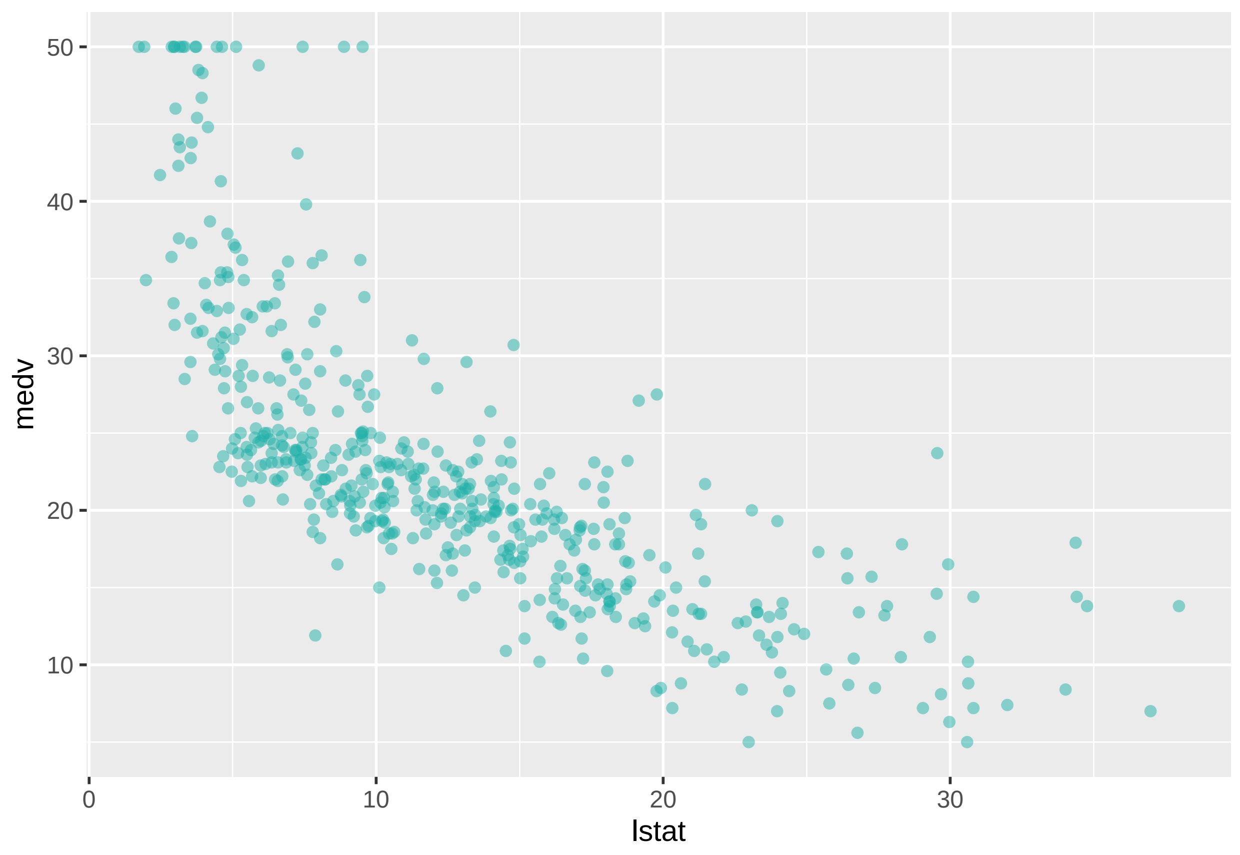
7. Use the newly created data frame as the newdata argument to a predict() call for lm_ses. Store it in a variable named y_pred_new.

```
y_pred_new = predict(lm.fit, pred_dat)
```

8. Create a scatter plot from the Boston dataset with lstat mapped to the x position and medv mapped to the y position. Store the plot in an object called p_scatter.

```
p_scatter <- ggplot(Boston) +
  geom_point(aes(x = lstat, y = medv), col = "Light Sea Green", alpha = 0.5)

p_scatter
```



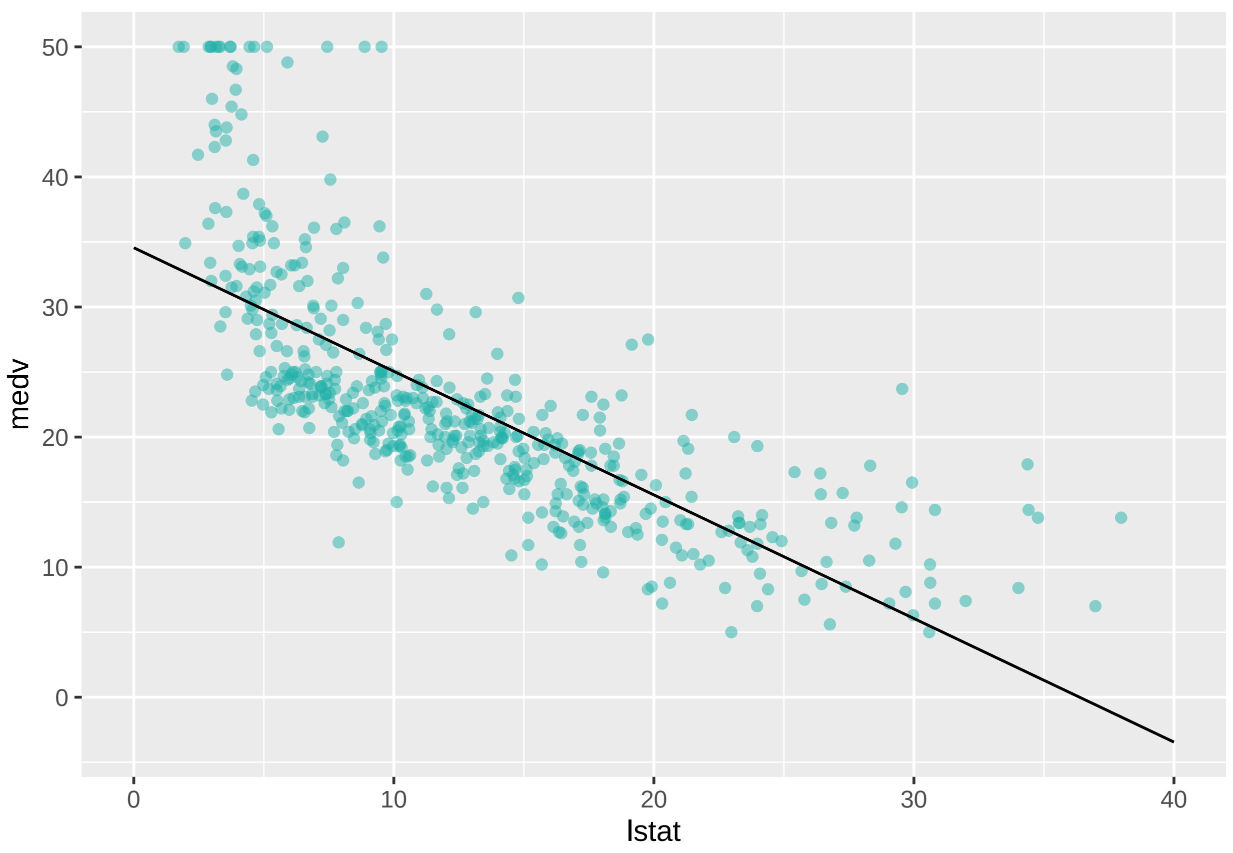
9. Add the vector y_pred_new to the pred_dat data frame with the name medv.

```
#cbind(pred_dat,y_pred_new)

pred_dat <- data.frame(lstat,y_pred_new) %>%
  mutate(mev = y_pred_new) %>%
  `colnames<-`(c('lstat', 'medv'))
```

10. Add a geom_line() to p_scatter, with pred_dat as the data argument. What does this line represent?

```
p_scatter +
  geom_line(data = pred_dat, aes(x = lstat, y = medv))
```



The 'black' line in the plot above shows the regression line for the predicted values for medv (using the lm_ses regression formula) against lstat.

11. The interval argument can be used to generate confidence or prediction intervals. Create a new object called y_pred_95 using predict() (again with the pred_dat data) with the interval argument set to "confidence". What is in this object?

```
lstat <- pred_dat$lstat

y_pred_95 <- predict(lm.fit, pred_dat, interval ="confidence")
```

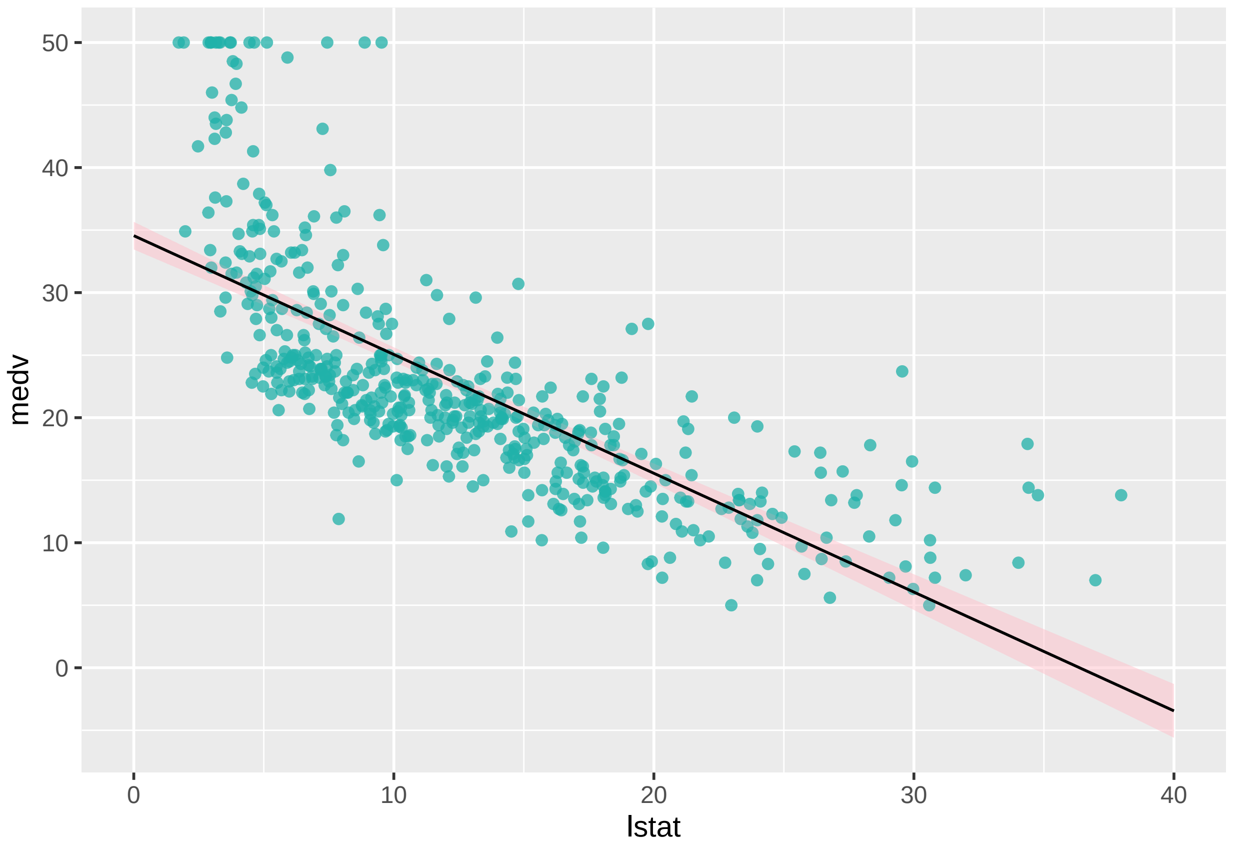
y_pred_95 is the confidence interval for each observation lstat in the pred_dat dataframe. It returns the fit, lower and upper bound as values.

12. Create a data frame with 4 columns: medv, lstat, lower, and upper.

```
confidence_dat <- cbind(pred_dat, y_pred_95)
```

13. Add a geom_ribbon() to the plot with the data frame you just made. The ribbon geom requires three aesthetics: x (lstat, already mapped), ymin (lower), and ymax (upper). Add the ribbon below the geom_line() and the geom_points() of before to make sure those remain visible. Give it a nice colour and clean up the plot, too!

```
p_scatter +
  geom_ribbon(data = confidence_dat, aes(x = lstat, ymin = lwr, ymax = upr), fill = "pi
  geom_point(aes(x = lstat, y = medv), col = "Light Sea Green", alpha = 0.5) +
  geom_line(data = pred_dat, aes(x = lstat, y = medv))
```



14. Explain in your own words what the ribbon represents.

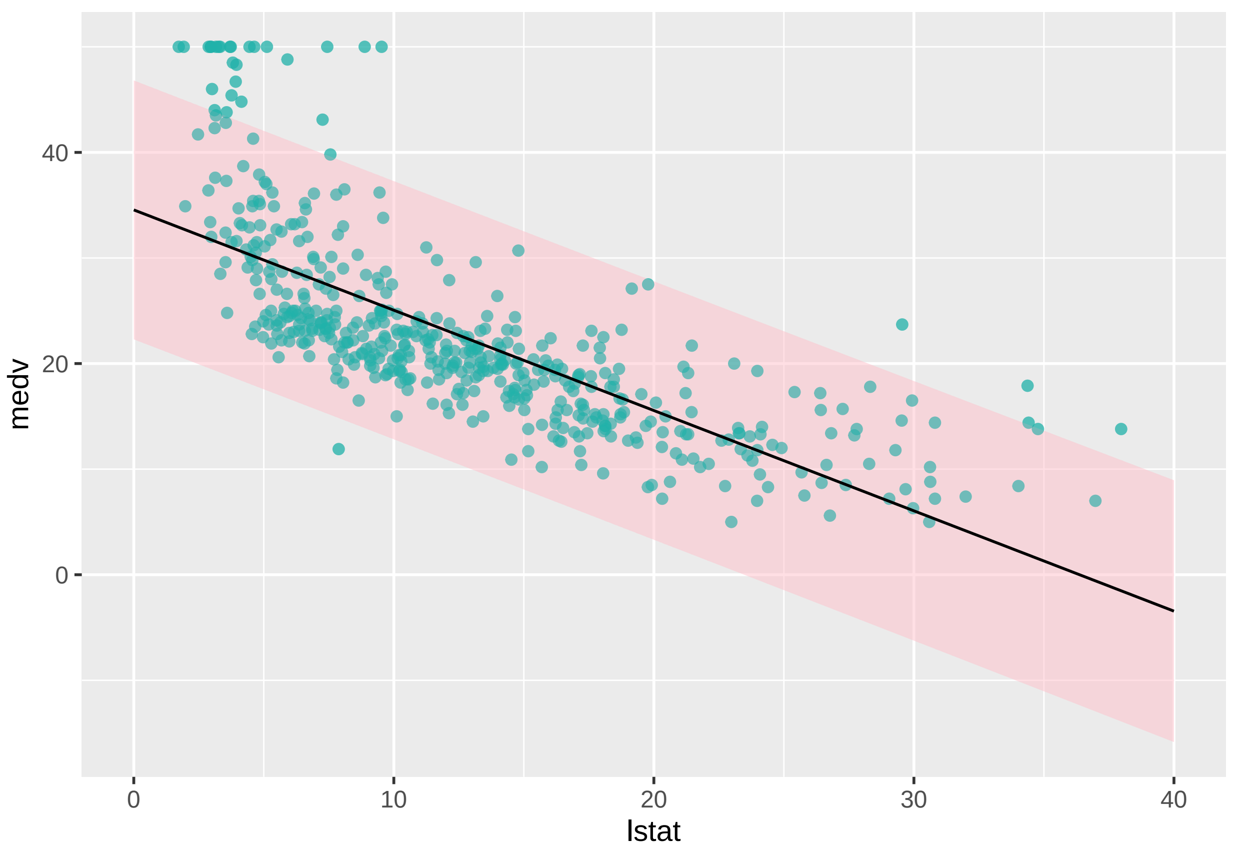
The ribbon represents the confidence bounds of which we expect with some level of certainty (often 95% confidence for example) that our fitted values will lie within a certain distance away from the fitted line.

15. Do the same thing, but now with the prediction interval instead of the confidence interval.

```
y_pred_95_2 <- predict(lm.fit, pred_dat, interval ="prediction")

prediction_dat <- cbind(pred_dat, y_pred_95_2)

p_scatter +
  geom_ribbon(data = prediction_dat, aes(x = lstat, ymin = lwr, ymax = upr), fill = "pi
  geom_point(aes(x = lstat, y = medv), col = "Light Sea Green", alpha = 0.5) +
  geom_line(data = pred_dat, aes(x = lstat, y = medv))
```



The ribbon represents the bounds for which a certain proportion of our data (often 95% of all data) we would expect to be found to lie.

16.

```r
mse <- function(y_true, y_pred) {
  (1/length(y_true))*sum((y_true - y_pred)^2)
}
```

17. Make sure your mse() function works correctly by running the following code.

```r
mse(1:10, 10:1)
```

```
## [1] 33
```

18. Calculate the mean square error of the lm_ses model. Use the medv column as y_true and use the predict() method to generate y_pred.

```r
mse(medv, y_pred)
```

```
## [1] 38.48297
```

19. The Boston dataset has 506 observations. Use c() and rep() to create a vector with 253 times the word "train", 152 times the word "validation", and 101 times the word "test". Call this vector splits.

```r
#train <- c(rep("train", 253))
#validation <- c(rep("validation", 152))
#test <- c(rep("test", 101))

splits <- c(rep("train", 253),rep("validation", 152), rep("test", 101))
```

20. Use the function sample() to randomly order this vector and add it to the Boston dataset using mutate(). Assign the newly created dataset to a variable called boston_master.

```r
boston_master <- Boston %>%
  mutate(splits=sample(splits))
```

21. Now use filter() to create a training, validation, and test set from the boston_master data. Call these datasets boston_train, boston_valid, and boston_test.

```r
boston_train <- filter(boston_master, splits == "train")
boston_validation <- filter(boston_master, splits == "validation")
boston_test <- filter(boston_master, splits == "test")
```

22. Train a linear regression model called model_1 using the training dataset. Use the formula medv ~ lstat like in the first lm() exercise. Use summary() to check that this object is as you expect.

```r
#fit a simple linear regression model
model_1 = lm(medv ~ lstat, data = boston_train)
summary(model_1)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = boston_train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.809 -4.014 -1.266  1.913 22.555
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.42852    0.78837   43.67   <2e-16 ***
## lstat       -0.93862    0.05467  -17.17   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.208 on 251 degrees of freedom
## Multiple R-squared:  0.5401, Adjusted R-squared:  0.5383
## F-statistic: 294.8 on 1 and 251 DF,  p-value: < 2.2e-16
```

23. Calculate the MSE with this object. Save this value as model_1_mse_train.

```
model_1_train <- predict(model_1, boston_train)

#add this column to existing data-frame
boston_train <- boston_train %>%
  mutate(train_pred = model_1_train)

model_1_mse_train <- mse(boston_train$medv, boston_train$train_pred)

model_1_mse_train
```

```
## [1] 38.23368
```

24. Now calculate the MSE on the validation set and assign it to variable model_1_mse_valid. Hint: use the newdata argument in predict().

```
#fit a simple linear regression model
model_1_valid <- predict(model_1, newdata = boston_validation)

#add this column to existing data-frame
boston_validation <- boston_validation %>%
  mutate(valid_pred = model_1_valid)

model_1_mse_valid <- mse(boston_validation$medv, boston_validation$valid_pred)

model_1_mse_valid
```

```
## [1] 40.51577
```

25. Create a second model model_2 for the train data which includes age and tax as predictors. Calculate the train and validation MSE.

```
#fit a multiple linear regression model
model_2 = lm(medv ~ lstat + age + tax, data = boston_train)

summary(model_2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + age + tax, data = boston_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.4922 -4.3187 -0.8539  2.1319 21.5537
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 35.777957   1.130445  31.649  < 2e-16 ***
## lstat       -0.870050   0.074253 -11.717  < 2e-16 ***
## age          0.025263   0.017228   1.466 0.143821
## tax         -0.009663   0.002823  -3.423 0.000723 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.083 on 249 degrees of freedom
## Multiple R-squared:  0.5619, Adjusted R-squared:  0.5566
## F-statistic: 106.5 on 3 and 249 DF,  p-value: < 2.2e-16
```

```
model_2_train <- predict(model_2, boston_train)

#add this column to existing data-frame
boston_train <- boston_train %>%
  mutate(train_pred = model_2_train)

model_2_mse_train <- mse(boston_train$medv, boston_train$train_pred)

model_2_mse_train
```

```
## [1] 36.42045
```

```
#fit a simple linear regression model
model_2_valid <- predict(model_2, newdata = boston_validation)

#add this column to existing data-frame
boston_validation <- boston_validation %>%
  mutate(valid_pred = model_2_valid)

model_2_mse_valid <- mse(boston_validation$medv, boston_validation$valid_pred)

model_2_mse_valid
```

```
## [1] 40.06131
```

26. Compare model 1 and model 2 in terms of their training and validation MSE. Which would you choose and why?

```
models_mse <- cbind(model_1_mse_train, model_1_mse_valid, model_2_mse_train, model_2_mse

models_mse
```

```
##      model_1_mse_train model_1_mse_valid model_2_mse_train model_2_mse_valid
## [1,]          38.23368          40.51577          36.42045          40.06131
```

We observe that the mean squared error is small on the training set under model_2. This can be explained by the fact that if we are indeed dealing with a non-linear model, then multiple-regression performs better than ordinary linear regression. However, when comes to newly unseen data the model_2 actually performed slightly worse (and proportinally much worse) than model_1.

27.Calculate the test MSE for the model of your choice in the previous question. What does this number tell you?

```r
#fit a simple linear regression model
model_2_test <- predict(model_2, newdata = boston_test)

#add this column to existing data-frame
boston_test <- boston_test %>%
  mutate(test_pred = model_2_test)

model_2_mse_test <- mse(boston_test$medv, boston_test$test_pred)

model_2_mse_test
```

```
## [1] 35.73995
```

We would like to find out if any further hints exist on whether our model is linear or not. We Therefore decided to test model_2 again. The mse is very large which suggest that we did not fit a very good model and perhaps we should not try using multiple regression methods by adding more parameters to the model fit, but instead keep it simple.

28. Create a function that performs k-fold cross-validation for linear models.

```r
#use same mse function created earlier
#mse <- function(y_true, y_pred) {
#  (1/length(y_true))*sum((y_true - y_pred)^2)

a <- 0
for (i in 1:9){
    set.seed(i)

    train = Boston %>%
      sample_frac(0.5) #p=0.5 because 253/506, but can also define length

    test = Boston %>%
      setdiff(train)

    # Fit a logistic regression to predict mdev
    model_logistic = glm(medv ~ lstat + age + tax, data = Boston)
    #model_logistic = glm(medv ~ lstat + I(lstat^2) + age + tax, data = Boston)

    # Use the model to predict the response on the test data
    # use train or test data for 'newdata' argument??
    model_logistic_val <- data.frame(val = predict(model_logistic, newdata = test))
    #add 'val' as column to existing data frame
    test = test %>%
      mutate(val = model_logistic_val)

    # calculate mean squared error
    a[i] <- mse(test$medv, test$val)
}

print(a)
```

```
## [1] 41.52239 34.24069 39.42158 40.86460 34.07584 39.90121 37.83435 34.76612
## [9] 36.37004
```

```r
kfold_sum_a <- sum(a)
average_a <- kfold_sum_a/9
average_a
```

```
## [1] 37.66631
```

29. Use your function to perform 9-fold cross validation with a linear model with as its formula medv ~ lstat + age + tax. Compare it to a model with as formulat medv ~ lstat + I(lstat^2) + age + tax.

```r
b <- 0
for (i in 1:9){
    set.seed(i)

    train = Boston %>%
      sample_frac(0.5) #p=0.5 because 253/506, but can also define length

    test = Boston %>%
      setdiff(train)

    # Fit a logistic regression to predict mdev
    #model_logistic = glm(medv ~ lstat + age + tax, data = Boston)
    model_logistic = glm(medv ~ lstat + I(lstat^2) + age + tax, data = Boston)

    # Use the model to predict the response on the test data
    # use train or test data for 'newdata' argument??
    model_logistic_val <- data.frame(val = predict(model_logistic, newdata = test))

    #add 'val' as column to existing data frame
    test = test %>%
      mutate(val = model_logistic_val)

    # calculate mean squared error
    b[i] <- mse(test$medv, test$val)
}

print(b)
```

```
## [1] 32.73094 25.47515 28.38075 29.96802 24.56833 28.24649 25.73574 25.15850
## [9] 27.96152
```

```
kfold_sum_b <- sum(b)
average_b <- kfold_sum_b/9
average_b
```

```
## [1] 27.5806
```

We observe from the answer in Q.28 that the average mse across the 9-fold cross validator is 37.66631 for the linear model. This is much larger than the average mse (27.5806) displayed in Q.29 for the linear model with a quadratic term form lstat. This would suggest using this quadratic term is a better fit for our linear regression model because the mse is minimized.