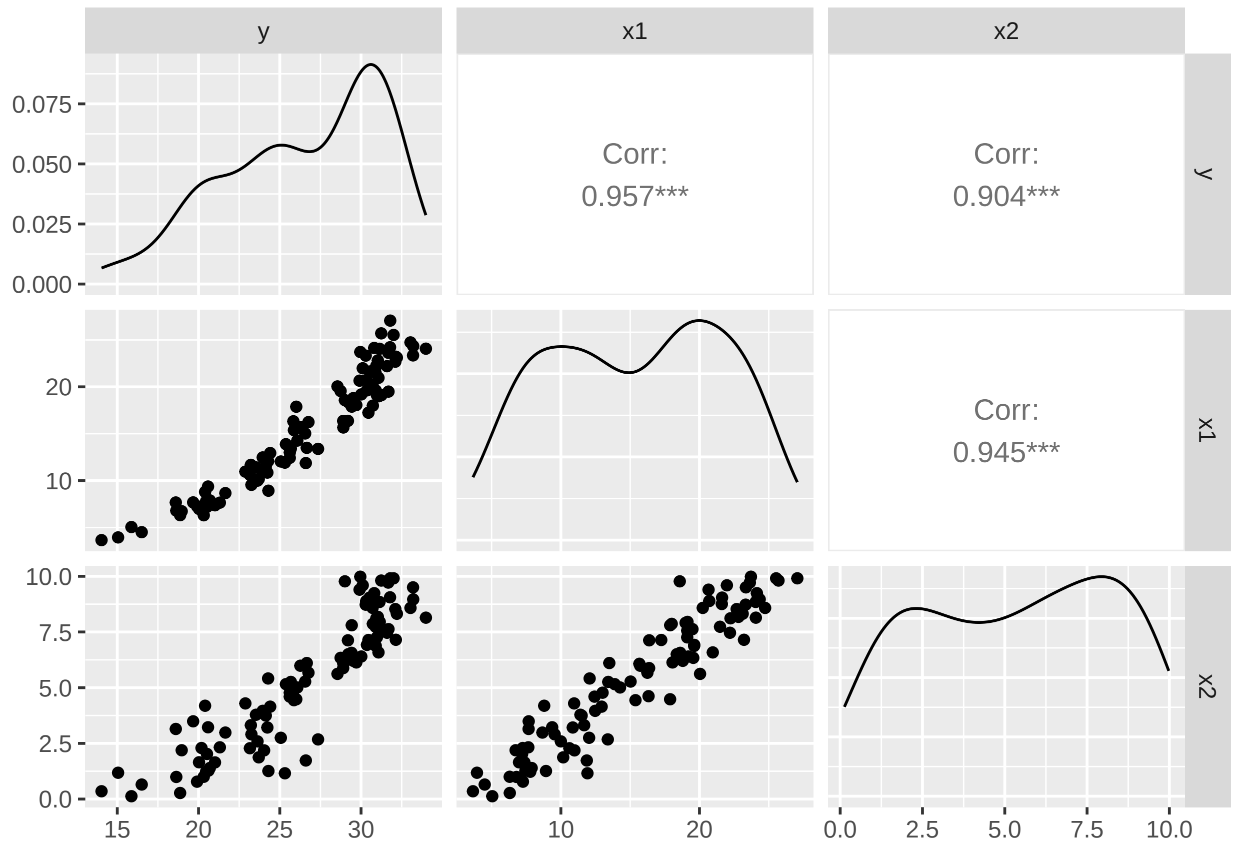
Coursework 2

30949

Regression and its limitations

Generate data [10 marks]

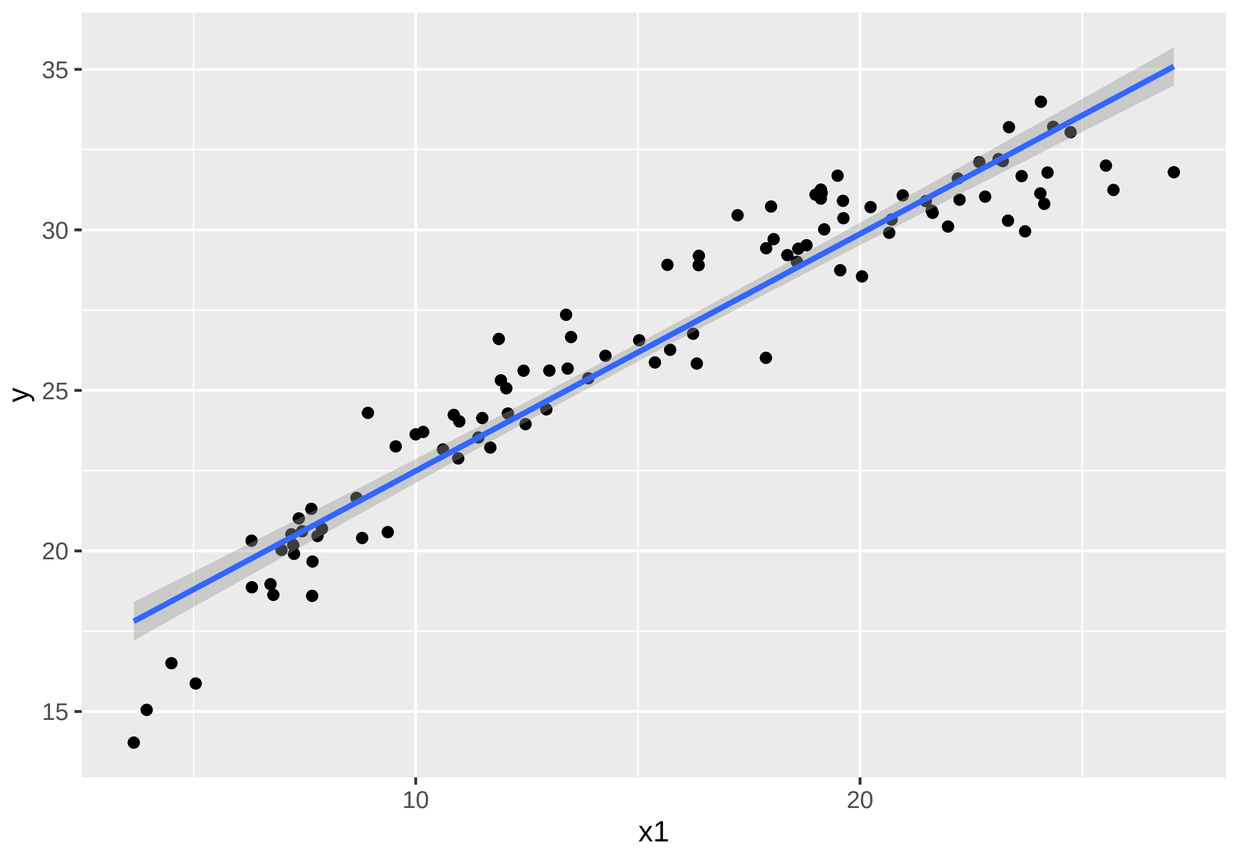
```
set.seed(30949)
n <- 100


x2 <- runif(n, min = 0, max = 10)
x1 <- 5 + 2 * x2 + rnorm(n, sd = 2)
X <- cbind(x1,x2)


Y_CEF <- function(x1, x2) 10*log(x1) + sin(x2)

simulate_data <- function(n, X) {
  x1 <- X[,1]
  x2 <- X[,2]
  errors <- rnorm(n)
  y <- Y_CEF(x1, x2) + errors
  data.frame(y = y,
             x1 = x1,
             x2 = x2)

}

sim_data <- simulate_data(n, cbind(X))
ggpairs(sim_data)
```



Explanation: We see that the data set has 2 predictor variables, x1 and x2. x2 is randomly generated from a uniform distribution, and is also a confounder of x1, as x1 is generated from a linear transformation of x2 with some error terms. The y values are then generated using a non-linear CEF. In the ggpairs plot, we see the logarithmic relationship between x1 and y quite clearly.

Linear model is a poor fit [15 marks]

```
linear_model <- lm(y ~ x1 + x2, data = sim_data)
scatterplot_lm <- ggplot(data=linear_model,aes(x=x1, y=y))+
  geom_point() +
  geom_smooth(method = "lm")
scatterplot_lm
```



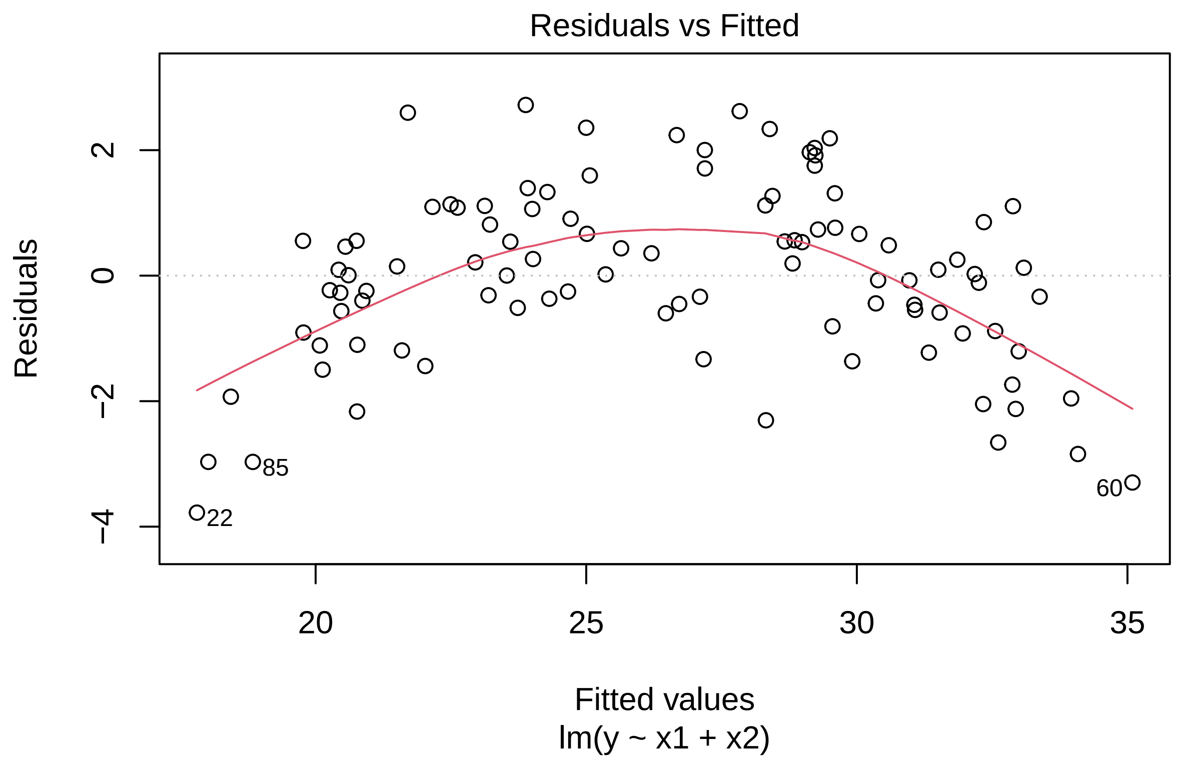
```
coef(linear_model)
```

```
##  (Intercept)           x1           x2
## 15.102905712  0.740279360 -0.004591075
```

```
X_svd <- svd(cbind(1,X))
X_pseudoinverse <- X_svd$v %*% solve(diag(X_svd$d)) %*% t(X_svd$u)
svd_estimate <- X_pseudoinverse %*% sim_data$y
t(svd_estimate)
```

```
##          [,1]      [,2]         [,3]
## [1,] 15.10291 0.7402794 -0.004591075
```

```
plot(linear_model, which = 1)
```



Explanation: We run a linear regression on both predictors using the simulated data. Comparing the coefficients that we get from this model to the ones we get from the SVD pseudoinverse, we see that they are identical. This is due to the fact that linear regression and the SVD pseudoinverse both work by minimising the sum of squared residuals.

Looking at the Fitted vs Residuals plot for the linear model, we notice the residuals follow a parabolic shape and are not randomly scattered around mean 0, suggesting there is still some signal in the data that has not been captured by the model.

Regression coefficient vs causal effect [15 marks]

```r
Y_new <- Y_CEF(X[,1] + 1, X[,2]) + rnorm(n)
dce <- mean(Y_new - sim_data$y)
dce
```

```
## [1] 0.9941143
```

```r
confint(linear_model, parm = "x1")
```

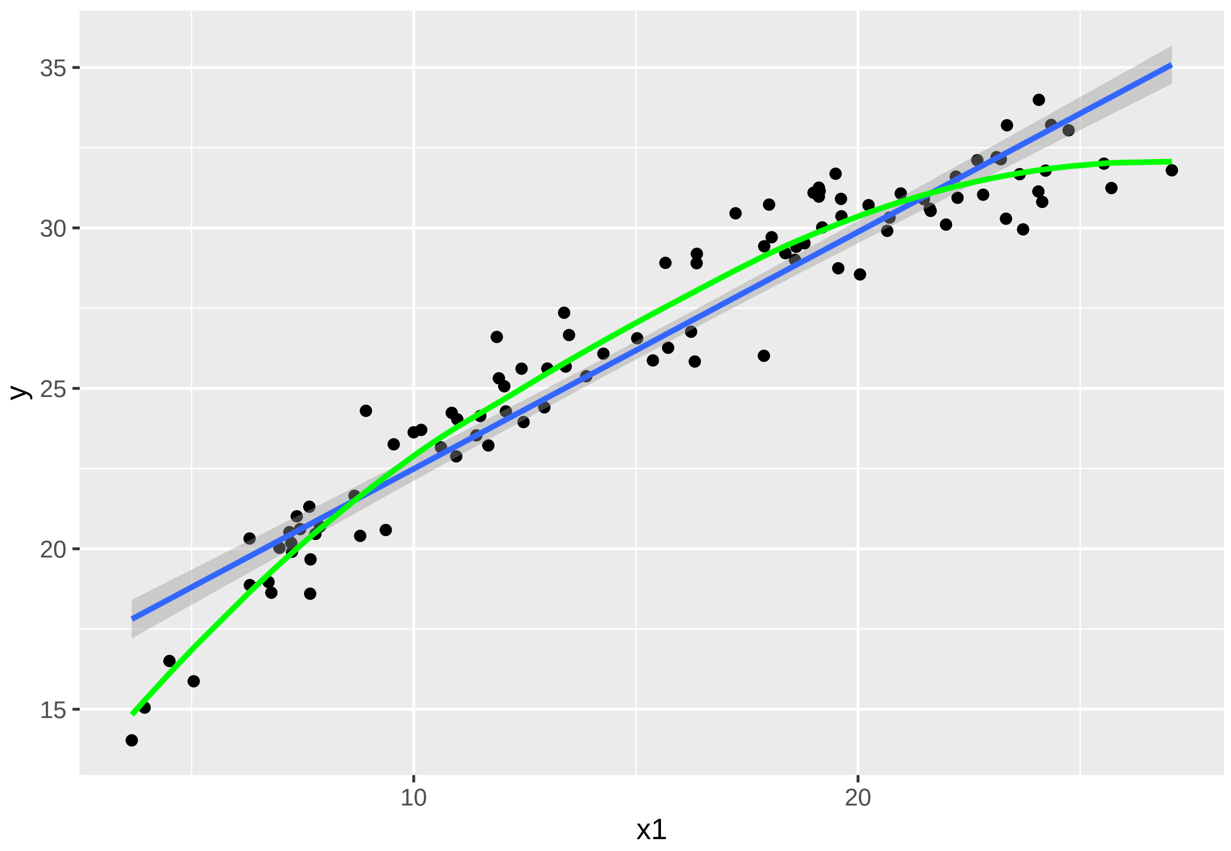
```
##        2.5 %    97.5 %
## x1 0.6010434 0.8795153
```

Explanation: We investigate the direct causal effect by increasing x1 by 1 unit, recalculating the y values, and working out the Average Treatment Effect. The interpretation of this is that if we increase x1 by 1 unit, we can expect y to increase by [dce] units on average.

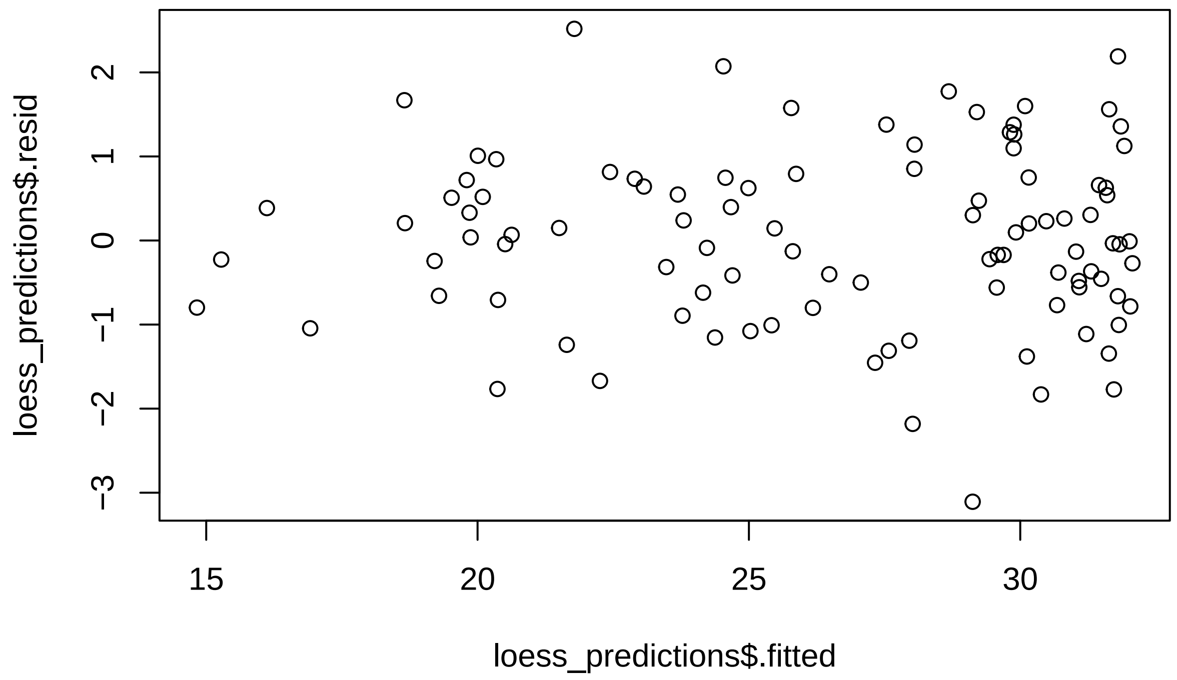
We also notice that the estimated causal effect is outside of the confidence interval for the estimate of the x1 coefficient. The interpretation of this confidence interval is that if we were to calculate the regression coefficient multiple times, it would fall within this interval 95% of the time. This suggests that there is sufficient evidence to suggest that the estimated causal effect is different than the effect estimated through the linear regression, which could be a result of the fact that we are attempting to use a linear regression to estimate a non-linear causal effect.

A good non-linear model [10 marks]

```
loess_model <- loess(y ~ x1, sim_data)
loess_predictions <- augment(loess_model)

scatterplot_lm +
  geom_line(data = loess_predictions, size = 1,
            color = "green",
            aes(x = x1, y = .fitted))
```



```
plot(x = loess_predictions$.fitted, y = loess_predictions$.resid)
```



Explanation: The loess model uses a local approximation method to derive predicted values that more accurately capture the logarithmic relationship between x1 and y. This improvement is evident in the fact that the Fitted vs Residuals plot now looks randomly scattered around 0 as opposed to having some parabolic pattern like before, suggesting a lot more of the signal has been captured by the model.

Causal relationship vs fitted [10 marks]

```
x2_new <- X[,2] + 1
x1_new <- 5 + 2 * x2_new + rnorm(n, sd = 2)
y_new <- Y_CEF(x1_new, x2_new) + rnorm(n)

tce <- mean(y_new - sim_data$y)
tce
```

```
## [1] 1.769123
```

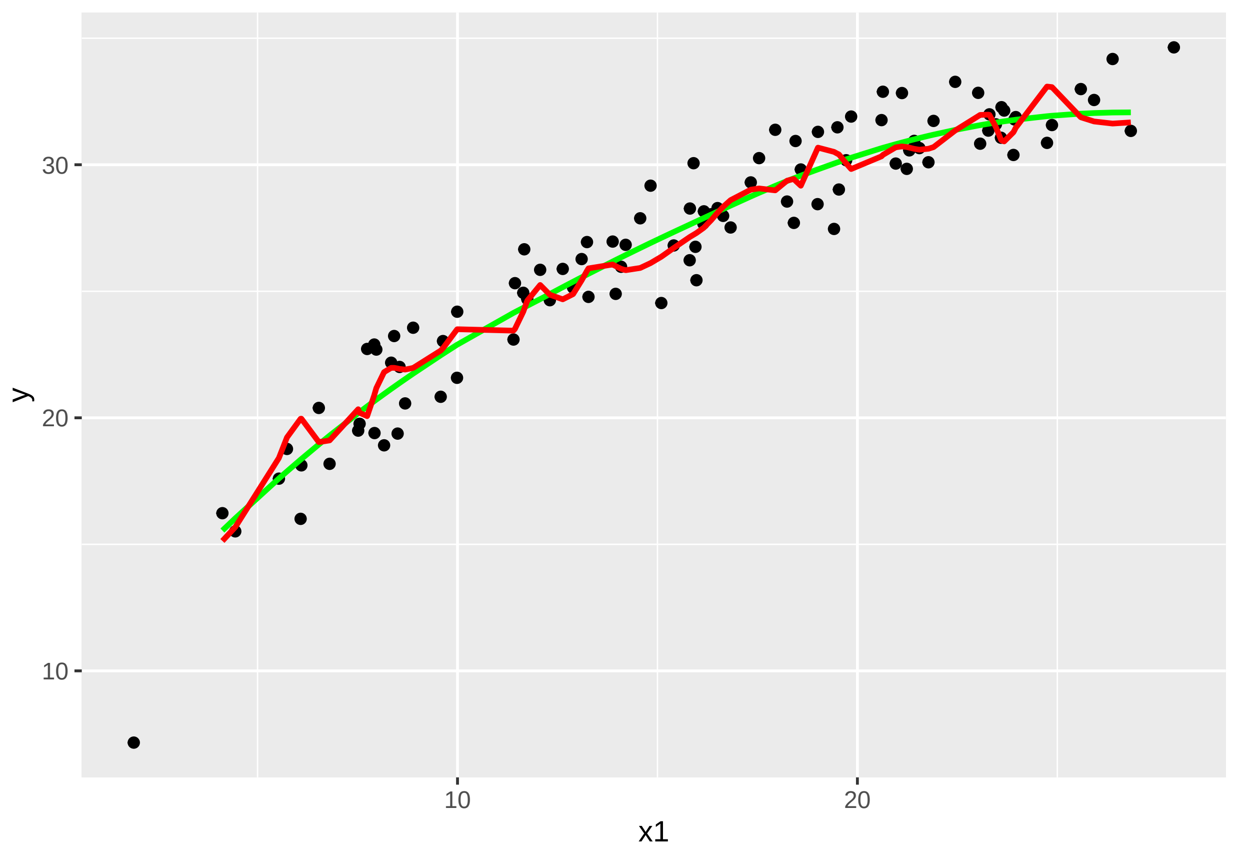
```
loess_predictions_sorted <- loess_predictions[order(loess_predictions$x1), ]
differences_fitted <- diff(loess_predictions_sorted$.fitted)
differences_x1 <- diff(loess_predictions_sorted$x1)
mean_gradient <- mean(differences_fitted / differences_x1)
mean_gradient
```

```
## [1] 0.7285439
```

Explanation: In this case, we are measuring the total effect, since we include both the direct effect of increasing x2 on y, as well as the indirect effect by propagating changes to x1 as well. We can interpret this as if we were to increase x2 by 1 unit, the total increase in y would be [tce] units. Since loess is a non-parametric model, there is no straightforward way to extract coefficients between predictor and response variables. What I chose to do instead is work out the mean gradient between consecutive predicted data points, and use this as an estimate for what the loess model suggests an increase unit in x1 would do to y. We can see that this mean gradient is significantly smaller than the Total Causal Effect, suggesting that the model doesn't fully capture this effect.

A bad non-linear model [10 marks]

```r
loess_model_bad <- loess(y ~ x1, sim_data, span = 0.1)

x2_predict <- runif(n, min = 0, max = 10)
x1_predict <- 5 + 2 * x2 + rnorm(n, sd = 2)
X_predict <- cbind(x1_predict,x2_predict)
sim_data_predict <- simulate_data(n, X_predict)

loess_predict_good_new <- augment(loess_model, newdata = sim_data_predict)
loess_predict_bad_new <- augment(loess_model_bad, newdata = sim_data_predict)
ggplot(data = sim_data_predict, mapping = aes(x = x1, y = y))+
  geom_point()+
  geom_line(data = loess_predict_good_new,
            mapping = aes(x = x1, y = .fitted),
            size = 1,
            colour = "green")+
  geom_line(data = loess_predict_bad_new,
            mapping = aes(x = x1, y = .fitted),
            size = 1,
            colour = "red")
```



```
mse_good <- na.omit(loess_predict_good_new) |>
  summarize(MSE_good = mean(.resid^2))
mse_bad <- na.omit(loess_predict_bad_new) |>
  summarize(MSE_bad = mean(.resid^2))
cbind(mse_good,mse_bad)
```

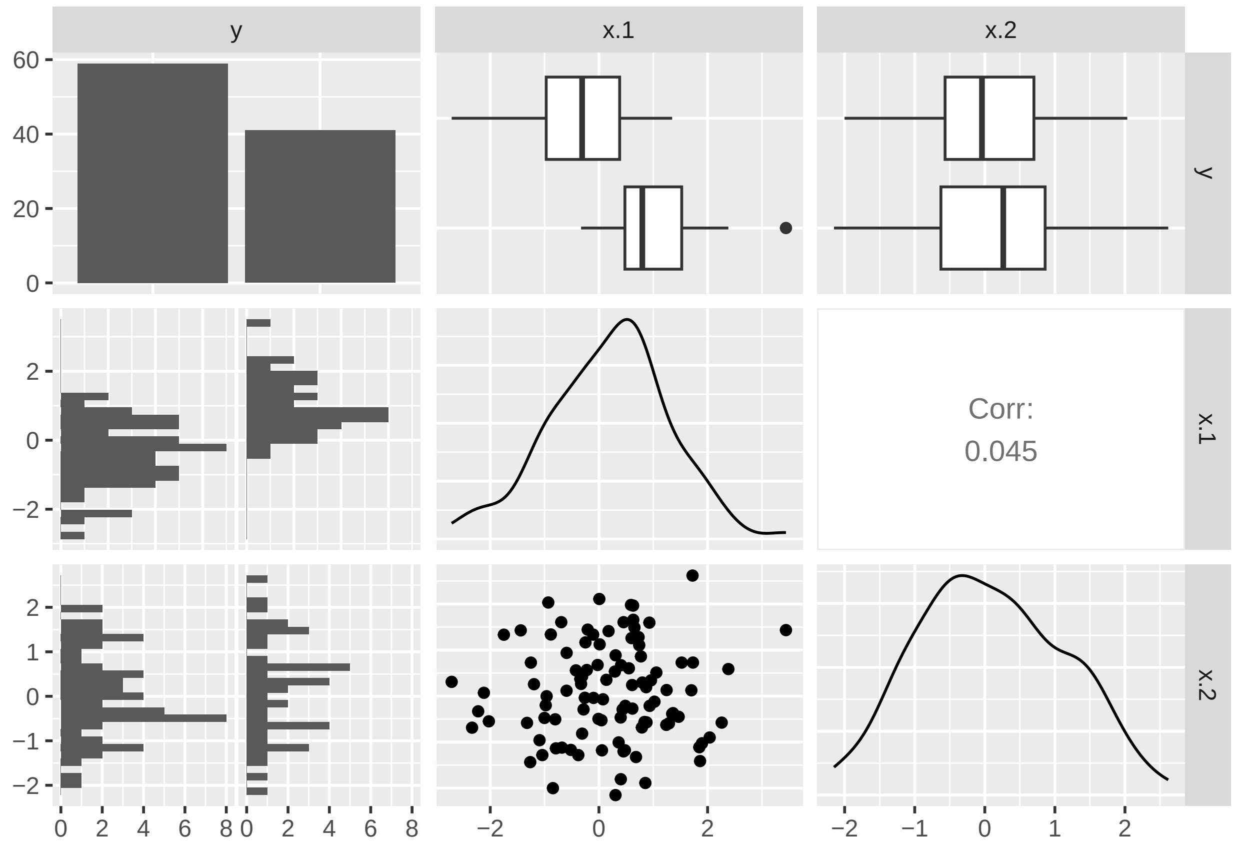
```
##   MSE_good  MSE_bad
## 1 1.575427 2.003932
```

Explanation: We make this model too complex by changing the span of the loess model to 0.1, which means it will use a more local estimation for each predicted value. This leads to the model overfitting on the training data, trying to include every individual point in its curve of best fit and missing the general relationship between x1 and y. As a result, when presented with a new set of data, the overfitted model has significantly worse prediction ability than the good model, as is evident in the good model's lower mean squared error.

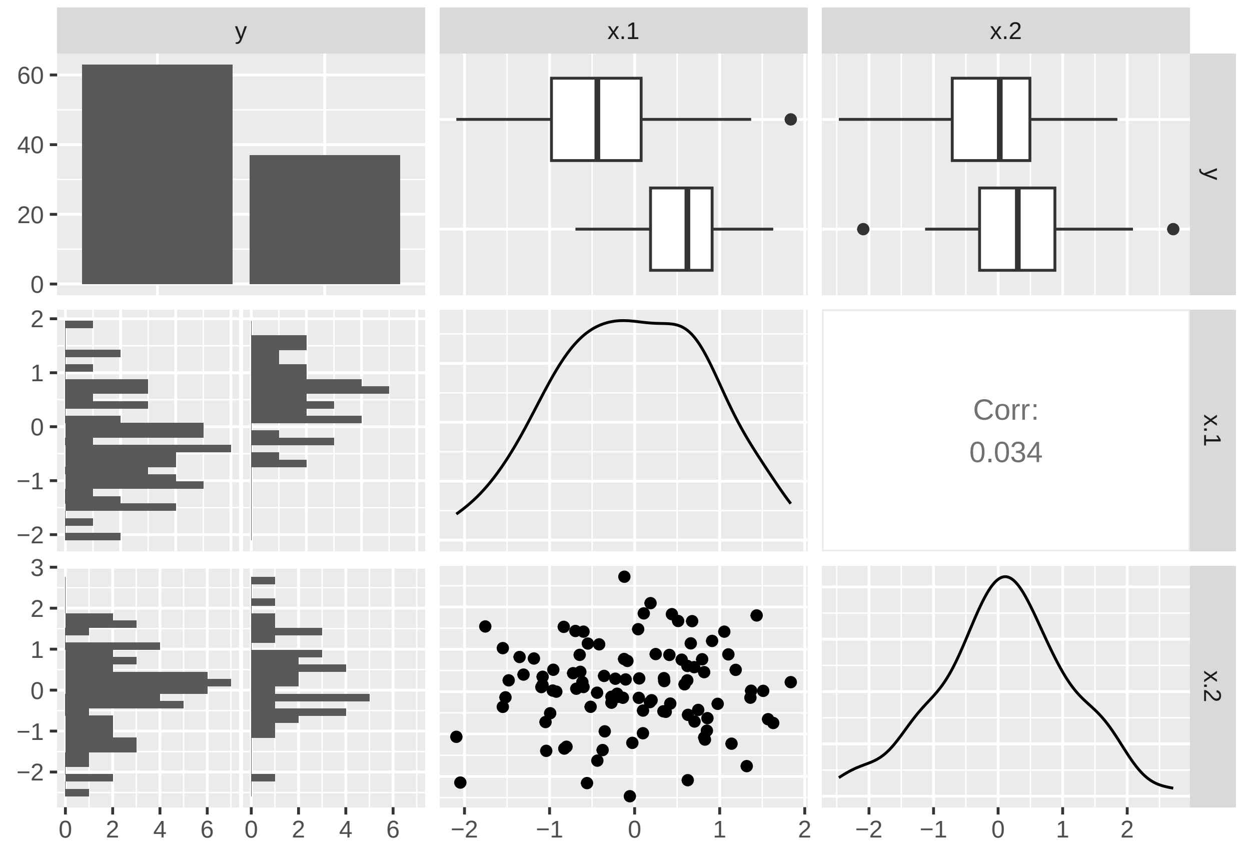
Open-ended question [20 marks]

- Investigate the effect of heteroscedasticity on a logistic regression model. Use two different sets of training data, one heteroskedastic and one not, and compare key performance measures for both models.

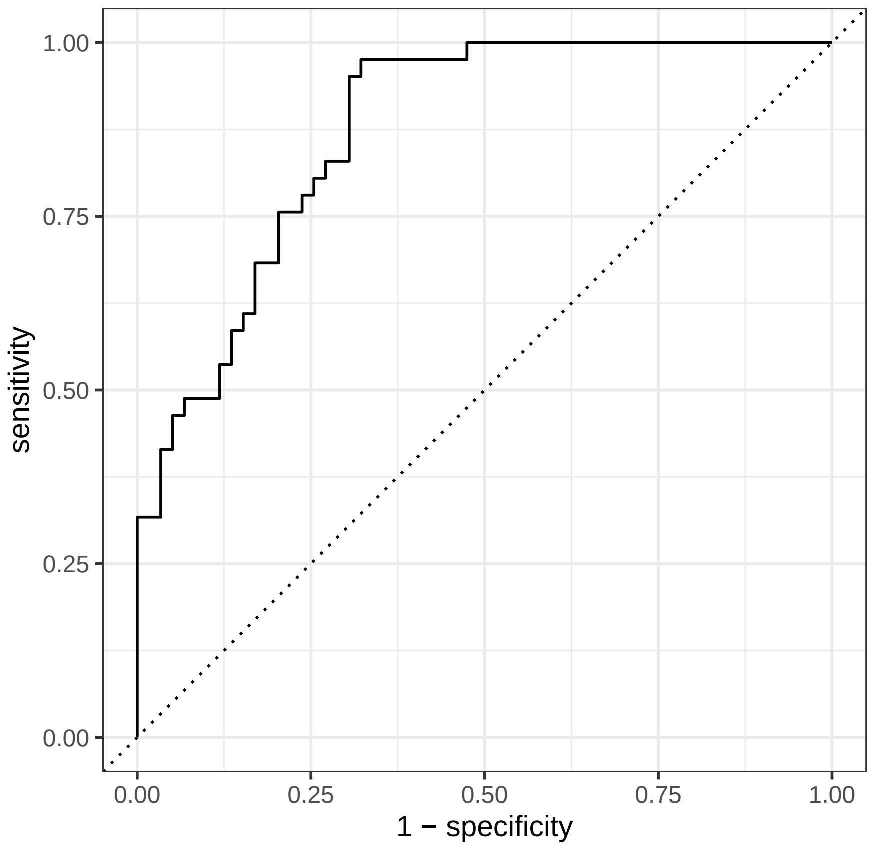
```r
n <- 100
X <- matrix(rnorm(2*n), nrow = n)
mu <- -1 + 2*X[,1] + 0.5*X[,2]
px <- exp(mu)/(1 + exp(mu))
Y <- rbinom(n, 1, prob = px)
lr_data_homo <- data.frame(y = as.factor(Y), x = X)
ggpairs(lr_data_homo)
```



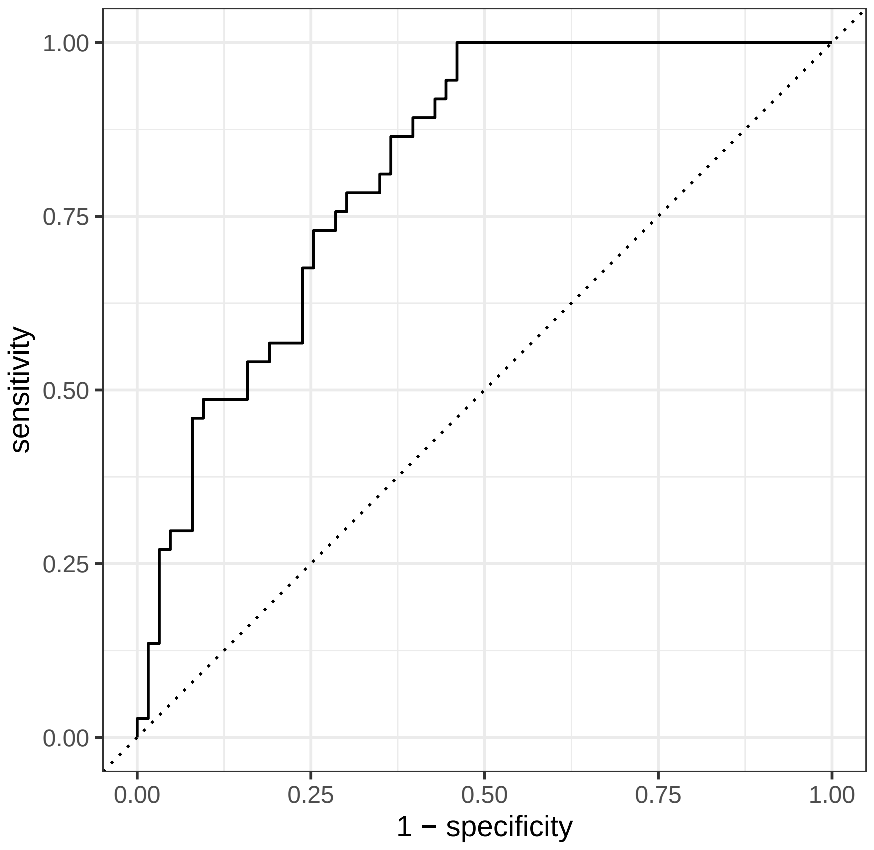
```r
yhat_success_homo <- mean(lr_data_homo$y == 1)

X <- matrix(rnorm(2*n), nrow = n)
error_sd <- abs(0.5*X[,1] + 0.2*X[,2])
errors <- rnorm(n, sd = error_sd)
mu <- -1 + 2*X[,1] + 0.5*X[,2] + errors
px <- exp(mu)/(1 + exp(mu))
Y <- rbinom(n, 1, prob = px)
lr_data_hetero <- data.frame(y = as.factor(Y), x = X)
lr_data_hetero <- na.omit(lr_data_hetero)
ggpairs(lr_data_hetero)
```



```r
yhat_success_hetero <- mean(lr_data_hetero$y == 1)

lr_model_homo <- glm(y ~ ., family = binomial(), data = lr_data_homo)
lr_predict_homo <- augment(lr_model_homo, type.predict = "response")
lr_predict_homo |>
  roc_curve(truth = y, .fitted,
            event_level = "second") |>
  autoplot()
```



```
lr_model_hetero <- glm(y ~ ., family = binomial(), data = lr_data_hetero)
lr_predict_hetero <- augment(lr_model_hetero, type.predict = "response")
lr_predict_hetero |>
  roc_curve(truth = y, .fitted,
            event_level = "second") |>
  autoplot()
```



```r
confusion_matrix_homo <-
  lr_predict_homo |>
  mutate(yhat = factor(as.numeric(.fitted > yhat_success_homo))) |>
  conf_mat(truth = y, estimate = yhat)

summary_homo <- summary(confusion_matrix_homo) |>
  mutate(homoscedastic = .estimate) |>
  select(.metric, homoscedastic)

confusion_matrix_hetero <-
  lr_predict_hetero |>
  mutate(yhat = factor(as.numeric(.fitted > yhat_success_hetero))) |>
  conf_mat(truth = y, estimate = yhat)

summary_hetero <- summary(confusion_matrix_hetero) |>
  mutate(heteroscedastic = .estimate) |>
  select(heteroscedastic)

cbind(summary_homo,summary_hetero)
```

```
##                 .metric homoscedastic heteroscedastic
## 1              accuracy     0.7600000       0.7400000
## 2                   kap     0.5147594       0.4603570
## 3                  sens     0.7457627       0.7460317
## 4                  spec     0.7804878       0.7297297
## 5                   ppv     0.8301887       0.8245614
## 6                  npv     0.6808511       0.6279070
## 7                  mcc     0.5185894       0.4639688
## 8              j_index     0.5262505       0.4757615
## 9         bal_accuracy     0.7631253       0.7378807
## 10 detection_prevalence     0.5300000       0.5700000
## 11            precision     0.8301887       0.8245614
## 12               recall     0.7457627       0.7460317
## 13               f_meas     0.7857143       0.7833333
```

Explanation: We investigate the effect of heteroscedasticity on logistic regression models by creating two different training data sets, one with heteroscedasticity and one with not. We do so by making the error term standard deviation be a function of the x values. We then create 2 models each trained on their corresponding training data set. We then set the prediction threshold to be equal to each data set's probability of success, in order to keep things controlled and make sure we are only measuring the effect of heteroscedasticity. Looking at the summary statistics for both models, we see that both models perform similarly on key statistics like accuracy, sensitivity and specificity. This leads us to the conclusion that heteroscedasticity does not have a significant impact on the performance of logistic regression models. This could be due to the fact that what really impacts the performance of a logistic classification is the classification threshold, since this determines how many data points the model correctly classifies. In this case, we kept the thresholds fixed at each model data set's respective probability of success, explaining the similar performance of both models.

References: Loftus (2021, Oct. 8). machine learning 4 data science: 4 Classification. Retrieved from http://ml4ds.com/weeks/04-classification/