```
# PairwiseCorrelations-IndependentVariables.R
# Computes Pearson and Spearman correlations and corresponding p-values between all possible pairs of variables
# that were considered as possible independent variables for our main multivariate regression model
# and as possible independent variables for the multivariate regressions in the sensitivity analyses
# Output written to spreadsheet PairwiseCorrelations-IndependentVariables.csv

fulldataset = read.csv(file="fulldataset.csv")

indepvariables =
fulldataset[,c("people_fully_vaccinated_per_hundred","gdp_per_capita","cardiovasc_death_rate","diabetes_prevalence","life_expectancy",
          "human_development_index","Stringency","aged_65_older","masks.avg","Obesity.rate","Gini","urban_density","total_tests_per_thousa
nd")]

var1 = character()
var2 = character()
cor_pearson = numeric()
pval_pearson = numeric()
cor_spearman = numeric()
pval_spearman = numeric()

line = 1
for(i in 1:(ncol(indepvariables)-1))
{
          for(j in (i+1):ncol(indepvariables))
          {
                    resultpearson = cor.test(indepvariables[,i],indepvariables[,j],method="pearson")
                    resultspearman = suppressWarnings(cor.test(indepvariables[,i],indepvariables[,j],method="spearman"))
                    var1[line] = names(indepvariables)[i]
                    var2[line] = names(indepvariables)[j]
                    cor_pearson[line] = resultpearson$estimate
                    pval_pearson[line] = resultpearson$p.val
                    cor_spearman[line] = resultspearman$estimate
                    pval_spearman[line] = resultspearman$p.val
                    line = line+1
          }
}

output =
data.frame(var1=var1,var2=var2,cor_pearson=cor_pearson,pval_pearson=pval_pearson,cor_spearman=cor_spearman,pval_spearman=pval_spearman)
output = output[order(abs(output$cor_pearson),decreasing=TRUE),]
write.csv(x=output,file="PairwiseCorrelations-IndependentVariables.csv",row.names=FALSE)

# CorrelationsWithPLAA.R
# Computes Pearson and Spearman correlations and corresponding p-values between the dependent variable PLAA and all variables
# that were considered as possible independent variables for our main multivariate regression model
# and as possible independent variables for the multivariate regressions in the sensitivity analyses
# cardiovascular death rate and life expectancy were substituted by cardlife, which is their principal component
# Output written to spreadsheet CorrelationsWithPLAA.csv

fulldataset = read.csv(file="fulldataset.csv")

# computes principal component of cardiovasc_death_rate and life_expectancy
prcomp_cardlife = prcomp(fulldataset[,c("cardiovasc_death_rate","life_expectancy")],scale=TRUE)
prcomp1_cardlife = prcomp_cardlife$x[,1] # extracts the principal component
coef_cardlife = prcomp_cardlife$rot[,1] # coefficients used for calculating the principal component
prop1_cardlife = prcomp_cardlife$sdev[1]^2/length(prcomp_cardlife$sdev) # proportion of total variance contained in the principal
component

indepvariables = fulldataset[,c("people_fully_vaccinated_per_hundred","gdp_per_capita","diabetes_prevalence",
          "human_development_index","Stringency","aged_65_older","masks.avg","Obesity.rate","Gini","urban_density","total_tests_per_thousa
nd")]
indepvariables$cardlife = prcomp1_cardlife

PLAA = fulldataset$Per.Levitt.Age.Adjusted

variable = colnames(indepvariables)
cor_pearson = numeric()
pval_pearson = numeric()
cor_spearman = numeric()
pval_spearman = numeric()

for(i in 1:ncol(indepvariables))
{
          resultpearson = cor.test(PLAA,indepvariables[,i],method="pearson")
          resultspearman = suppressWarnings(cor.test(PLAA,indepvariables[,i],method="spearman"))
          cor_pearson[i] = resultpearson$estimate
          pval_pearson[i] = resultpearson$p.val
          cor_spearman[i] = resultspearman$estimate
          pval_spearman[i] = resultspearman$p.val
}

output =
data.frame(variable=variable,cor_pearson=cor_pearson,pval_pearson=pval_pearson,cor_spearman=cor_spearman,pval_spearman=pval_spearman)
write.csv(x=output,file="CorrelationsWithPLAA.csv",row.names=FALSE)

# MainRegression.R
# Computes the main regression and sensitivity analyses based on bootstrap and robust sandwich standard errors
# Computes all diagnostics for the main regression
# Outputs: MainRegression.txt; summary of main regression (coefficients and standardized coefficients, 95% confidence intervals and p-
values)
# Sensitivity.txt; sensitivity analyses of main regression using bootstrap and heterocedastic-consistent sandwich standard errors
# VariousDiagnostics.txt; multicolinearity check, skewness, excess kurtosis and Shapiro-Wilk p-value for standardized residuals, Breusch-
Pagan test for homocedasticity
# fittedresidual.jpg; scatter plot of fitted values versus residuals
# fittedstdresidual.jpg; scatter plot of fitted values versus standardized residuals
# scalelocation.jpg; scale-location plot (fitted values versus square root of absolute values of standardized residuals)
# qqnormal.jpg; qq plot of standardized residuals against normal distribution
# cooks_distance.jpg; bar plot of Cook's distances
```

```
library(ggplot2)
library(car)
library(sandwich)
library(DescTools)
library(lmtest)
library(betaDelta)
library(betaSandwich)

bootreg = function(y,X,nrep=10^5,conf.level=0.95){
# computes multivariable linear regression coefficients and standardized coefficients
# plus corresponding confidence intervals and p-values using bootstrap
# y = dependent variable;
# X = independent variables (should be a matrix and should not include a column of 1's for intercept, which is automatically included)
# nrep = number of bootstraps; conf.level = confidence level for confidence intervals
# returns a list with fields coef (coefficients), IClwr, ICupr (confidence interval)
# std_coef (standardized coefficients), std_IClwr, std_ICupr (confidence interval), pval (p-value)

sestdcoefs = function(y,X,coefs,sigma2){
# auxiliary function for function bootreg
# computes standardized regression coefficients (stdcoef) and corresponding standard errors (se)
# X should not include a column of 1's for intercept
# coefs = (nonstandardized) coefficients for regression excluding intercept
# sigma2 = estimate of residual variance

          X = as.matrix(X)
          SX = var(X)
          SXy = apply(X,2,function(u) cov(u,y))
          Sy2 = var(y)
          n = length(y)
          stdcoef = coefs*sqrt(diag(SX))/sqrt(Sy2)
          SXinv = solve(SX)
          se2 = diag(SX)*diag(SXinv)*sigma2/((n-3)*Sy2) + coefs^2*(diag(SX)*as.vector(coefs%*%SX%*%coefs)-diag(SX)*sigma2-SXy^2)/((n-
3)*Sy2^2)
          return(list(stdcoef=stdcoef,se=sqrt(se2)))
}

oneboot = function(y,X,maincoefs,mainstdcoefs){
# auxiliary function for function bootreg
# does one iteration of bootstrap, returns "t-values" for coefficients and standardized coefficients
# where "t-value" equals the coefficient from bootstrapped sample minus coefficient from main data divided by estimate of standard error
# a column of 1's must be included as first column of X for intercept

linreg = function(y,X,maincoefs,mainstdcoefs){
# does the main calculation for function oneboot

          covunsc = solve(t(X)%*%X)
          coefs = covunsc%*%t(X)%*%y
          resid = y-X%*%coefs
          sigma2 = sum(resid^2)/(nrow(X)-ncol(X))
          se = sqrt(diag(covunsc))*sqrt(sigma2)
          tval = (coefs-maincoefs)/se
          calcstdcoefs = sestdcoefs(y,X[,-1],coefs[-1],sigma2)
          stdtval = (calcstdcoefs$stdcoef-mainstdcoefs)/calcstdcoefs$se
          return(c(tval,stdtval))
}

          samp = sample.int(nrow(X),nrow(X),replace=TRUE) # bootstrap sample
          return(linreg(y[samp],X[samp,],maincoefs,mainstdcoefs))
}

          # computes coefficients, standardized coefficients, standard errors and t-values for main data
          reg = summary(lm(y~X))
          maincoefs = reg$coefficients[,1]
          mainse = reg$coefficients[,2]
          maintvals = reg$coefficients[,3]
          calcstdcoefs = sestdcoefs(y,X,maincoefs[-1],reg$sigma^2)
          mainstdcoefs = calcstdcoefs$stdcoef
          mainsestd = calcstdcoefs$se

          X = cbind(1,X)
          # computes bootstrapped "t-values"; first ncol(X) lines of samptvals for regular coefficients (includes intercept)
          # last ncol(X)-1 lines of samptvals for standardized coefficients (excludes intercept)
          samptvals = matrix(replicate(nrep,oneboot(y,X,maincoefs,mainstdcoefs)),nrow=2*ncol(X)-1)

          pvals = sapply(1:ncol(X),function(i) sum(abs(samptvals[i,]) >= abs(maintvals[i]))/nrep)
          q = apply(samptvals,1,function(u) c(quantile(u,(1-conf.level)/2,na.rm=TRUE),quantile(u,(1+conf.level)/2,na.rm=TRUE))) #
quantiles from bootstraps
          return(list(coef=maincoefs,IClwr=maincoefs-mainse*q[2,1:ncol(X)],ICupr=maincoefs-
mainse*q[1,1:ncol(X)],std_coef=c(NA,mainstdcoefs),
                    std_IClwr=c(NA,mainstdcoefs-mainsestd*q[2,(ncol(X)+1):(2*ncol(X)-1)]),std_ICupr=c(NA,mainstdcoefs-mainsestd*q[1,
(ncol(X)+1):(2*ncol(X)-1)]),pval=pvals))
}

bootregression = function(resultreg,nrep=10^5,conf.level=0.95){
# like function bootreg, but instead of receiving y and X as input
# receives resultreg, which is the output of an lm command (regression must include intercept)

          return(bootreg(resultreg$model[,1],as.matrix(resultreg$model[,-1]),nrep,conf.level))
}

HCregression = function(resultreg,conf.level=0.95){
# computes multivariable linear regression coefficients and standardized coefficients
# plus corresponding confidence intervals and p-values using heterocedastic-consistent sandwich standard errors (method HC3)
# resultreg = output of an lm command; conf.level = confidence level for confidence intervals
# returns a list with fields coef (coefficients), IClwr, ICupr (confidence interval)
# std_coef (standardized coefficients), std_IClwr, std_ICupr (confidence interval), pval (p-value)

          se = sqrt(diag(vcovHC(resultreg,type="HC3")))
          pval = 2*pt(abs(resultreg$coefficients)/se,df=resultreg$df,lower.tail=FALSE)
```

```
	radius = se*qt((1+conf.level)/2,df=resultreg$df)
	stdreg = BetaHC(resultreg,type="hc3")
	stdcoef = c(NA,coef(stdreg))
	IC = confint(stdreg,level=conf.level)
	return(list(coef=resultreg$coefficients,IClwr=resultreg$coefficients-radius,ICupr=resultreg$coefficients+radius,
		std_coef=stdcoef,std_IClwr=c(NA,IC[,1]),std_ICupr=c(NA,IC[,2]),pval=pval))
}

stdresiduals = function(resultreg){
# computes standardized regression residuals
# resultreg = output of an lm command

	h = lm.influence(resultreg)$hat
	return(resultreg$residuals/(sqrt(1-h)*summary(resultreg)$sigma))
}

checkmulticol = function(resultreg){
# multicolinearity check
# computes R squared for the regression of each independent variable against all others
# resultreg = output of an lm command (regression must include intercept)

	X = as.matrix(resultreg$model[,-1])
	if(ncol(X) == 1)
	{
		return(NULL)
	}
	R2 = numeric(ncol(X))
	names(R2) = colnames(X)
	for(i in 1:ncol(X))
	{
		y = X[,i]
		X1 = X[,-i]
		R2[i] = summary(lm(y~X1))$r.squared
	}
	return(R2)
}

predictedR2 = function(resultreg){
# computes predicted R squared
# resultreg = output of an lm command

	h = lm.influence(resultreg)$hat
	PRESS = sum((resultreg$residuals/(1-h))^2)
	y = resultreg$model[,1]
	TSS = sum((y-mean(y))^2)
	return(1-PRESS/TSS)
}

summaryreg = function(resultreg,varnames=names(resultreg$coefficients)[-1],conf.level=0.95){
# computes multivariable linear regression coefficients and standardized coefficients
# plus corresponding confidence intervals and p-values
# resultreg = output of an lm command; varnames = names of independent variables (excluding intercept)
# conf.level = confidence level for confidence intervals
# returns a data frame with fields coef (coefficients), IClwr, ICupr (confidence interval)
# std_coef (standardized coefficients), std_IClwr, std_ICupr (confidence interval), pval (p-value)

	coef = resultreg$coefficients
	IClwr = coef-summary(resultreg)$coefficients[,2]*qt((1+conf.level)/2,df=resultreg$df)
	ICupr = coef+summary(resultreg)$coefficients[,2]*qt((1+conf.level)/2,df=resultreg$df)
	stdreg = BetaDelta(resultreg,type="mvn")
	stdcoef = coef(stdreg)
	stdIC = confint(stdreg,level=conf.level)
	result = data.frame(coef=coef,IClwr=IClwr,ICupr=ICupr,
		std_coef=c(NA,stdcoef),std_IClwr=c(NA,stdIC[,1]),std_ICupr=c(NA,stdIC[,2]),pval=summary(resultreg)$coefficients[,4])
	rownames(result) = c("Intercept",varnames)
	return(result)
}

output_summaryreg = function(resultreg,varnames=names(resultreg$coefficients)[-1],conf.level=0.95){
# generates text output for function summaryreg, includes adjusted and predicted R squared
# resultreg = output of an lm command; varnames = names of independent variables (excluding intercept)
# conf.level = confidence level for confidence intervals

	ncols = getOption("width")
	options(width=300)
	output = capture.output(summaryreg(resultreg,varnames,conf.level))
	output = c(output,"",paste("Adjusted R squared:",summary(resultreg)$adj.r.squared),paste("Predicted R
squared:",predictedR2(resultreg)))
	options(width=ncols)
	return(output)
}

output_bootregression = function(resultreg,varnames=names(resultreg$coefficients)[-1],nrep=10^5,conf.level=0.95){
# generates text output for function bootregression
# resultreg = output of an lm command (regression must include intercept); varnames = names of independent variables (excluding intercept)
# nrep = number of bootstraps; conf.level = confidence level for confidence intervals

	result = data.frame(bootregression(resultreg,nrep,conf.level))
	rownames(result) = c("Intercept",varnames)

	ncols = getOption("width")
	options(width=300)
	output = capture.output(result)
	options(width=ncols)
	return(output)
}

output_HCregression = function(resultreg,varnames=names(resultreg$coefficients)[-1],conf.level=0.95){
# generates text output for function HCregression
# resultreg = output of an lm command (regression must include intercept); varnames = names of independent variables (excluding intercept)
# conf.level = confidence level for confidence intervals

          result = data.frame(HCregression(resultreg,conf.level))
          rownames(result) = c("Intercept",varnames)

          ncols = getOption("width")
          options(width=300)
          output = capture.output(result)
          options(width=ncols)
          return(output)
}

plot_fittedresidual = function(resultreg,abbreviation=NA){
# generates plot of fitted values versus residuals
# resultreg = output of an lm command; abbreviation = labels for points

          if(is.na(abbreviation[1]))
          {
                    abbreviation = rep("",nrow(resultreg$model))
          }
          dataplot = data.frame(labels=abbreviation,x=resultreg$fitted.values,y=resultreg$residuals)
          return(ggplot(dataplot,aes(x=x,y=y,label=labels))+geom_point()+geom_text(vjust=1.5)
                    +labs(x="fitted values",y="residuals")+stat_smooth(formula=y~x,method="loess"))
}

plot_fittedstdresidual = function(resultreg,abbreviation=NA){
# generates plot of fitted values versus standardized residuals
# resultreg = output of an lm command; abbreviation = labels for points

          if(is.na(abbreviation[1]))
          {
                    abbreviation = rep("",nrow(resultreg$model))
          }
          dataplot = data.frame(labels=abbreviation,x=resultreg$fitted.values,y=stdresiduals(resultreg))
          return(ggplot(dataplot,aes(x=x,y=y,label=labels))+geom_point()+geom_text(vjust=1.5)
                    +labs(x="fitted values",y="standardized residuals")+stat_smooth(formula=y~x,method="loess"))
}

plot_scalelocation = function(resultreg,abbreviation=NA){
# generates scale-location plot (fitted values versus square root of absolute values of standardized residuals)
# resultreg = output of an lm command; abbreviation = labels for points

          if(is.na(abbreviation[1]))
          {
                    abbreviation = rep("",nrow(resultreg$model))
          }
          dataplot = data.frame(labels=abbreviation,x=resultreg$fitted.values,y=sqrt(abs(stdresiduals(resultreg))))
          return(ggplot(dataplot,aes(x=x,y=y,label=labels))+geom_point()+geom_text(vjust=1.5)
                    +labs(x="fitted values",y="sqrt of abs value of standardized residuals")+stat_smooth(formula=y~x,method="loess"))
}

plot_qqnormal = function(resultreg){
# generates qq plot of standardized residuals against a normal distribution
# resultreg = output of an lm command

          dataplot = data.frame(y=stdresiduals(resultreg))
          return(ggplot(dataplot,aes(sample=y))+stat_qq()+stat_qq_line()+labs(x="theoretical quantile",y="standardized residuals"))
}

plot_cook = function(resultreg,abbreviation=NA){
# generates bar plot of Cook's distances
# resultreg = output of an lm command; abbreviation = labels for bars

          if(is.na(abbreviation[1]))
          {
                    abbreviation = rep("",nrow(resultreg$model))
          }
          dataplot = data.frame(labels=abbreviation,y=cooks.distance(resultreg))
          return(ggplot(dataplot,aes(x=labels,y=y))+labs(x="countries",y="Cook's distance")+geom_bar(stat="identity"))
}

# beginning of main code

fulldataset = read.csv(file="fulldataset.csv")

# computes principal component of cardiovasc_death_rate and life_expectancy
prcomp_cardlife = prcomp(fulldataset[,c("cardiovasc_death_rate","life_expectancy")],scale=TRUE)
prcomp1_cardlife = prcomp_cardlife$x[,1] # extracts the principal component
coef_cardlife = prcomp_cardlife$rot[,1] # coefficients used for calculating the principal component
prop1_cardlife = prcomp_cardlife$sdev[1]^2/length(prcomp_cardlife$sdev) # proportion of total variance contained in the principal
component

# data for main regression
regdata = fulldataset[,c("people_fully_vaccinated_per_hundred","human_development_index","masks.avg","Per.Levitt.Age.Adjusted")]
regdata$cardlife = prcomp1_cardlife

mainreg = lm(Per.Levitt.Age.Adjusted~people_fully_vaccinated_per_hundred+human_development_index+cardlife+masks.avg,data=regdata)

output = output_summaryreg(mainreg)
outputfile = file("MainRegression.txt")
writeLines(output,outputfile)
close(outputfile)

separator = paste(rep("=",40),collapse="")

set.seed(123) # for reproducibility of bootstrap
output = c("Bootstrap:",output_bootregression(mainreg),separator,"Heterocedastic-consistent sandwich standard
erros:",output_HCregression(mainreg))
```

```
outputfile = file("Sensitivity.txt")
writeLines(output,outputfile)
close(outputfile)

output = c("R2 of regression of each independent variable against all other independent
variables:",capture.output(checkmulticol(mainreg)))
output = c(output,separator,"Skewness for standardized
residuals:",capture.output(Skew(stdresiduals(mainreg),method=2,conf.level=0.95,ci.type="bca",R=10^5)))
output = c(output,separator,"Excess kurtosis for standardized
residuals:",capture.output(Kurt(stdresiduals(mainreg),method=2,conf.level=0.95,ci.type="bca",R=10^5)))
output = c(output,separator,"Shapiro-Wilk for standardized residuals:",capture.output(shapiro.test(stdresiduals(mainreg))))
output = c(output,separator,"Breusch-Pagan test for homocedasticity:",capture.output(bptest(mainreg,studentize=TRUE)))
outputfile = file("VariousDiagnostics.txt")
writeLines(output,outputfile)
close(outputfile)

plot_fittedresidual(mainreg,fulldataset$abbreviation)
ggsave(filename="fittedresidual.jpg",device="jpeg",width=14.23,height=6.77)

plot_fittedstdresidual(mainreg,fulldataset$abbreviation)
ggsave(filename="fittedstdresidual.jpg",device="jpeg",width=14.23,height=6.77)

plot_scalelocation(mainreg,fulldataset$abbreviation)
ggsave(filename="scalelocation.jpg",device="jpeg",width=14.23,height=6.77)

plot_qqnormal(mainreg)
ggsave(filename="qqnormal.jpg",device="jpeg",width=14.23,height=6.77)

plot_cook(mainreg,fulldataset$abbreviation)
ggsave(filename="cooks_distance.jpg",device="jpeg",width=14.23,height=6.77)

# OtherRegressions.R
# Computes all possible regressions obtained by adding at most 3 extra independent variables to the 4 independent variables considered in
the main regression
# computes coefficients, standardized coefficients, 95% confidence intervals and p-values
# Outputs: OtherRegressions.txt; text file with results of all regressions
# OtherRegressions.csv; spreadsheet with results of all regressions

library(ggplot2)
library(car)
library(betaDelta)

predictedR2 = function(resultreg){
# computes predicted R squared
# resultreg = output of an lm command

          h = lm.influence(resultreg)$hat
          PRESS = sum((resultreg$residuals/(1-h))^2)
          y = resultreg$model[,1]
          TSS = sum((y-mean(y))^2)
          return(1-PRESS/TSS)
}

summaryreg = function(resultreg,varnames=names(resultreg$coefficients)[-1],conf.level=0.95){
# computes multivariable linear regression coefficients and standardized coefficients
# plus corresponding confidence intervals and p-values
# resultreg = output of an lm command; varnames = names of independent variables (excluding intercept)
# conf.level = confidence level for confidence intervals
# returns a data frame with fields coef (coefficients), IClwr, ICupr (confidence interval)
# std_coef (standardized coefficients), std_IClwr, std_ICupr (confidence interval), pval (p-value)

          coef = resultreg$coefficients
          IClwr = coef-summary(resultreg)$coefficients[,2]*qt((1+conf.level)/2,df=resultreg$df)
          ICupr = coef+summary(resultreg)$coefficients[,2]*qt((1+conf.level)/2,df=resultreg$df)
          stdreg = BetaDelta(resultreg,type="mvn")
          stdcoef = coef(stdreg)
          stdIC = confint(stdreg,level=conf.level)
          result = data.frame(coef=coef,IClwr=IClwr,ICupr=ICupr,
                    std_coef=c(NA,stdcoef),std_IClwr=c(NA,stdIC[,1]),std_ICupr=c(NA,stdIC[,2]),pval=summary(resultreg)$coefficients[,4])
          rownames(result) = c("Intercept",varnames)
          return(result)
}

output_summaryreg = function(resultreg,varnames=names(resultreg$coefficients)[-1],conf.level=0.95){
# generates text output for function summaryreg, includes adjusted and predicted R squared
# resultreg = output of an lm command; varnames = names of independent variables (excluding intercept)
# conf.level = confidence level for confidence intervals

          ncols = getOption("width")
          options(width=300)
          output = capture.output(summaryreg(resultreg,varnames,conf.level))
          output = c(output,"",paste("Adjusted R squared:",summary(resultreg)$adj.r.squared),paste("Predicted R
squared:",predictedR2(resultreg)))
          options(width=ncols)
          return(output)
}

# beginning of main code

fulldataset = read.csv(file="fulldataset.csv")

# computes principal component of cardiovasc_death_rate and life_expectancy
prcomp_cardlife = prcomp(fulldataset[,c("cardiovasc_death_rate","life_expectancy")],scale=TRUE)
prcomp1_cardlife = prcomp_cardlife$x[,1] # extracts the principal component
coef_cardlife = prcomp_cardlife$rot[,1] # coefficients used for calculating the principal component
prop1_cardlife = prcomp_cardlife$sdev[1]^2/length(prcomp_cardlife$sdev) # proportion of total variance contained in the principal
component

# four variables that must be included as independent variables in the sensitivity analyses regressions
```

```
mustinclude = fulldataset[,c("masks.avg","people_fully_vaccinated_per_hundred","human_development_index")]
mustinclude$cardlife = prcomp1_cardlife
mustinclude = as.matrix(mustinclude)

# other variables that can be included as independent variables in the sensitivity analyses regressions - at most 3 at a time
others =
as.matrix(fulldataset[,c("Stringency","total_tests_per_thousand","aged_65_older","urban_density","diabetes_prevalence","Obesity.rate","Gin
i")])

PLAA = fulldataset$Per.Levitt.Age.Adjusted

# generates data frame corresponding to all possible subsets of size at most 3 from the "other" variables
possibilities = expand.grid(replicate(ncol(others),c(0,1),simplify=FALSE))
possibilities = possibilities[apply(possibilities,1,sum)<=3,]

outputsheet = matrix(nrow=nrow(possibilities),ncol=4+(ncol(mustinclude)+ncol(others))*7+2)
colnamessheet = c("Intercept","Intercept_IClwr","Intercept_ICupr","Intercept_pval")
colnamessheet = c(colnamessheet,
          unlist(lapply(c(colnames(mustinclude),colnames(others)),
          function(x) sapply(c("","_IClwr","_ICupr","_std","_std_IClwr","_std_ICupr","_pval"),function(y) paste(x,y,sep="")))))
colnamessheet = c(colnamessheet,"adj_R2","pred_R2")
colnames(outputsheet) = colnamessheet

separator = paste(rep("=",40),collapse="")

legend_textfile = c("Legend:","coef: coefficient value","IClwr: lower extremity of 95% confidence interval for coefficient",
          "ICupr: upper extremity of 95% confidence interval for coefficient",
          "std_coef: standardized coefficient","std_IClwr: lower extremity of 95% confidence interval for standardized coefficient",
          "std_ICupr: upper extremity of 95% confidence interval for standardized coefficient","pval: p-value")

legend_sheet = c("Legend:","Column with variable name (varname) alone is the coefficient",
          "varname_IClwr is the lower extremity of 95% confidence interval for the coefficient",
          "varname_ICupr is the upper extremity of 95% confidence interval for the coefficient",
          "varname_std is the standardized coefficient",
          "varname_std_IClwr is the lower extremity of the 95% confidence interval for the standardized coefficient",
          "varname_std_ICupr is the upper extremity of the 95% confidence interval for the standardized coefficient",
          "varname_pval is the p-value",
          "adj_R2 is the adjusted R squared",
          "pred_R2 is the predicted R squared")

outputtext = c()
for(i in 1:nrow(possibilities))
{
          X = cbind(mustinclude,others[,possibilities[i,]==1])
          regression = lm(PLAA~X)
          sumreg = summaryreg(regression)
          sheetline = unlist(sumreg[1,c("coef","IClwr","ICupr","pval")])
          sheetline = c(sheetline,unlist(lapply(1:ncol(mustinclude),function(j) sumreg[j+1,])))
          k = ncol(mustinclude)+1
          for(j in 1:ncol(others))
          {
                    if(possibilities[i,j] == 1)
                    {
                              k = k+1
                              sheetline = c(sheetline,unlist(sumreg[k,]))
                    }else
                    {
                              sheetline = c(sheetline,rep(NA,7))
                    }
          }
          sheetline = c(sheetline,summary(regression)$adj.r.squared,predictedR2(regression))
          outputsheet[i,] = sheetline
          varnames = c(colnames(mustinclude),colnames(others)[possibilities[i,]==1])
          outputtext = c(outputtext,output_summaryreg(regression,varnames),separator)
}

outputfile = file("OtherRegressions.txt")
writeLines(c(legend_textfile,separator,outputtext),outputfile)
close(outputfile)

write.csv(x=data.frame(outputsheet),file="OtherRegressions.csv",na="",row.names=FALSE)
outputfile = file("OtherRegressions.csv","a")
writeLines(c("",legend_sheet),outputfile)
close(outputfile)

# PowerCalculation.R
# Power calculations for sensitivity analyses regressions
# uses main regression as "pilot study" for obtaning estimates of true values of relevant parameters
# Output: PowerCalculation.txt; table with power calculations for detecting a nonzero coefficient for masks
# as a function of the number of independent variables included in the regression

powerlinreg = function(n,p,beta,Risq,Rsq,sig.level=0.05){
# power calculation for linear regression Y=beta_0+beta_1X_1+...+beta_pX_p+epsilon
# testing the null hypothesis beta_i=0 (for some fixed i in {1,...,p})
# n = sample size, p = number of regressors
# beta = beta_i sd(X_i)/sd(Y) = standardized regression coefficient for X_i
# Risq = R-squared when X_i is regressed against all other X_j
# Rsq = 1 - Var(epsilon)/Var(Y) = R-squared when Y is regressed against X_1, ..., X_p
# sig.level = significance level (two-tailed p-value is considered)
# power calculation is exact if X_1, ..., X_p are regarded as a fixed nonrandom sample of size n;
# for exactness to hold, Risq should be the sample adjusted R-squared and sd(X_i) should be the sqrt of the unbiased sample variance of
X_i

          df = n-p-1
          tcrit = qt(sig.level/2,df=df,lower.tail=FALSE)
          ncp = sqrt(1-Risq)*sqrt(n-p)*beta/sqrt(1-Rsq)
          return(pt(-tcrit,df=df,ncp=ncp)+pt(tcrit,df=df,ncp=ncp,lower.tail=FALSE))
}
```

```
sampleadjR2var = function(nrep,n,p,Rsq,varY){
# creates a sample of size nrep from the distribution of adjusted R squared and the unbiased sample variance of Y
# assuming Y is a sample of size n of Y=beta_0+beta_1X_1+...+beta_pX_p+epsilon, with epsilon normal with mean zero
# X_1, ..., X_p multivariate normal; epsilon independent of (X_1,...,X_p)
# varY = true variance of Y
# Rsq = true R squared = var(beta_1X_1+...+beta_pX_p)/var(Y)

          if(p == 0)
          {
                    chi = rchisq(nrep,df=n-1)
                    result = rbind(0,chi*varY*(1-Rsq)/(n-1))
          }else
          {
                    chi1 = rchisq(nrep,df=n-p-1)
                    chi2 = rchisq(nrep,df=p,ncp=Rsq/(1-Rsq)*rchisq(nrep,df=n-1))
                    sumchi = chi1+chi2
                    result = rbind(1-(chi1/sumchi)*((n-1)/(n-p-1)),sumchi*varY*(1-Rsq)/(n-1))
          }
          row.names(result) = c("adjRsq","hatvarY")
          return(result)
}

powerlinreg_randomX = function(n,p,beta,Risq,Rsq,nrep=10^5,sig.level=0.05){
# power calculation for linear regression Y=beta_0+beta_1X_1+...+beta_pX_p+epsilon based on Monte Carlo simulation
# nrep = number of simulations
# testing the null hypothesis beta_i=0 (for some fixed i in {1,...,p})
# n = sample size, p = number of regressors
# beta = beta_i sd(X_i)/sd(Y) = standardized regression coefficient for X_i
# Risq = true R-squared when X_i is regressed against all other X_j
# Rsq = 1 - Var(epsilon)/Var(Y) = true R-squared when Y is regressed against X_1, ..., X_p
# sig.level = significance level (two-tailed p-value is considered)
# power calculation is exact if X_1, ..., X_p is multivariate normal

          simulation = sampleadjR2var(nrep,n,p-1,Risq,1)
          # note that function sampleadjR2var is being used to simulate regressions of X_i against all other X_j (not regression of Y
against all X_i)
          # thus simulation["hatvarY",] is a simulation of sample variance of X_i and
          # simulation["adjRsq",] is a simulation of adjusted R squared of regression of X_i against all other X_j

          return(mean(Vectorize(powerlinreg)(n,p,beta*sqrt(simulation["hatvarY",]),simulation["adjRsq",],Rsq,sig.level)))
}

# beginning of main code

set.seed(123) # for reproducibility

fulldataset = read.csv(file="fulldataset.csv")

# computes principal component of cardiovasc_death_rate and life_expectancy
prcomp_cardlife = prcomp(fulldataset[,c("cardiovasc_death_rate","life_expectancy")],scale=TRUE)
prcomp1_cardlife = prcomp_cardlife$x[,1] # extracts the principal component
coef_cardlife = prcomp_cardlife$rot[,1] # coefficients used for calculating the principal component
prop1_cardlife = prcomp_cardlife$sdev[1]^2/length(prcomp_cardlife$sdev) # proportion of total variance contained in the principal
component

# data for main regression
regdata = fulldataset[,c("people_fully_vaccinated_per_hundred","human_development_index","masks.avg","Per.Levitt.Age.Adjusted")]
regdata$cardlife = prcomp1_cardlife

mainreg = lm(Per.Levitt.Age.Adjusted~people_fully_vaccinated_per_hundred+human_development_index+cardlife+masks.avg,data=regdata)

# standardized coefficient for masks.avg in main regression
betamain = mainreg$coefficients["masks.avg"]*sd(regdata$masks.avg)/sd(regdata$Per.Levitt.Age.Adjusted)

# adjusted R squared for regression of masks.avg against other independent variables of main regression
Risqmain = summary(lm(masks.avg~people_fully_vaccinated_per_hundred+human_development_index+cardlife,data=regdata))$adj.r.squared

# adjusted R squared for main regression
Rsqmain = summary(mainreg)$adj.r.squared

separator = paste(rep("=",40),collapse="")

output = c("Parameter estimates from main regression:")
output = c(output,paste("Standardized coefficient for masks:",betamain))
output = c(output,paste("Adjusted R2 of regression of masks against other independent variables:",Risqmain))
output = c(output,paste("Adjusted R2 of main regression",Rsqmain))

n = nrow(regdata)
beta = 0.5
Risq = 0.4
Rsq = 0.75
output = c(output,separator)
output = c(output,"Parameters used for power calculation:")
output = c(output,paste("Sample size:",n),paste("True standardized coefficient for masks:",beta))
output = c(output,paste("true R2 of regression of masks against other independent variables:",Risq))
output = c(output,paste("true R2 of regression of PLAA against independent variables:",Rsq))
output = c(output,"Power estimates (0.05 significance level):")
output = c(output,capture.output(print(data.frame("number of variables"=4:10,power=Vectorize(powerlinreg_randomX)
(n=n,p=4:10,beta=beta,Risq=Risq,Rsq=Rsq)))))
outputfile = file("PowerCalculation.txt")
writeLines(output,outputfile)
close(outputfile)

# SimulationPlots.R
# Generates diagnostic plots for linear regressions based on simulated data satisfying linear model assumptions exactly
# used for comparisons witih diagnostic plots from real data
# number of simulations = 10
# Outputs: simulated-fittedresidual.pdf; fitted values versus residuals
# simulated-fittedstdresidual.pdf; fitted values versus standardized residuals
```

```r
# simulated-scalelocation.pdf; scale-location plots (fitted values versus squared roots of absolute values of standardized residuals)
# simulated-qqnormal.pdf; qq plots of standardized residuals against normal distribution

library(ggplot2)

stdresiduals = function(resultreg){
# computes standardized regression residuals
# resultreg = output of an lm command

         h = lm.influence(resultreg)$hat
         return(resultreg$residuals/(sqrt(1-h)*summary(resultreg)$sigma))
}

plot_fittedresidual = function(resultreg,abbreviation=NA){
# generates plot of fitted values versus residuals
# resultreg = output of an lm command; abbreviation = labels for points

         if(is.na(abbreviation[1]))
         {
                  abbreviation = rep("",nrow(resultreg$model))
         }
         dataplot = data.frame(labels=abbreviation,x=resultreg$fitted.values,y=resultreg$residuals)
         return(ggplot(dataplot,aes(x=x,y=y,label=labels))+geom_point()+geom_text(vjust=1.5)
                  +labs(x="fitted values",y="residuals")+stat_smooth(formula=y~x,method="loess"))
}

plot_fittedstdresidual = function(resultreg,abbreviation=NA){
# generates plot of fitted values versus standardized residuals
# resultreg = output of an lm command; abbreviation = labels for points

         if(is.na(abbreviation[1]))
         {
                  abbreviation = rep("",nrow(resultreg$model))
         }
         dataplot = data.frame(labels=abbreviation,x=resultreg$fitted.values,y=stdresiduals(resultreg))
         return(ggplot(dataplot,aes(x=x,y=y,label=labels))+geom_point()+geom_text(vjust=1.5)
                  +labs(x="fitted values",y="standardized residuals")+stat_smooth(formula=y~x,method="loess"))
}

plot_scalelocation = function(resultreg,abbreviation=NA){
# generates scale-location plot (fitted values versus square root of absolute values of standardized residuals)
# resultreg = output of an lm command; abbreviation = labels for points

         if(is.na(abbreviation[1]))
         {
                  abbreviation = rep("",nrow(resultreg$model))
         }
         dataplot = data.frame(labels=abbreviation,x=resultreg$fitted.values,y=sqrt(abs(stdresiduals(resultreg))))
         return(ggplot(dataplot,aes(x=x,y=y,label=labels))+geom_point()+geom_text(vjust=1.5)
                  +labs(x="fitted values",y="sqrt of abs value of standardized residuals")+stat_smooth(formula=y~x,method="loess"))
}

plot_qqnormal = function(resultreg){
# generates qq plot of standardized residuals against a normal distribution
# resultreg = output of an lm command

         dataplot = data.frame(y=stdresiduals(resultreg))
         return(ggplot(dataplot,aes(sample=y))+stat_qq()+stat_qq_line()+labs(x="theoretical quantile",y="standardized residuals"))
}

# beginning of main code

fulldataset = read.csv(file="fulldataset.csv")

# computes principal component of cardiovasc_death_rate and life_expectancy
prcomp_cardlife = prcomp(fulldataset[,c("cardiovasc_death_rate","life_expectancy")],scale=TRUE)
prcomp1_cardlife = prcomp_cardlife$x[,1] # extracts the principal component
coef_cardlife = prcomp_cardlife$rot[,1] # coefficients used for calculating the principal component
prop1_cardlife = prcomp_cardlife$sdev[1]^2/length(prcomp_cardlife$sdev) # proportion of total variance contained in the principal
component

# data for main regression
regdata = fulldataset[,c("people_fully_vaccinated_per_hundred","human_development_index","masks.avg","Per.Levitt.Age.Adjusted")]
regdata$cardlife = prcomp1_cardlife

mainreg = lm(Per.Levitt.Age.Adjusted~people_fully_vaccinated_per_hundred+human_development_index+cardlife+masks.avg,data=regdata)
sigma = summary(mainreg)$sigma # residual variance estimate from main regression
X = as.matrix(mainreg$model[,-1]) # matrix with independent variables from main regression

nsimul = 10 # number of simulations

set.seed(123) # for reproducibility

# generates simulated regressions
simulatedregs = list()
for(i in 1:nsimul)
{
         simulatedy = cbind(1,X)%*%mainreg$coefficients+rnorm(nrow(regdata),0,sigma)
         simulatedregs[[i]] = lm(simulatedy~X)
}

pdf("simulated-fittedresidual.pdf",width=14.23,height=6.77)
for(i in 1:nsimul)
{
         print(plot_fittedresidual(simulatedregs[[i]],NA))
}
dev.off()

pdf("simulated-fittedstdresidual.pdf",width=14.23,height=6.77)
```

```
for(i in 1:nsimul)
{
          print(plot_fittedstdresidual(simulatedregs[[i]],NA))
}
dev.off()

pdf("simulated-scalelocation.pdf",width=14.23,height=6.77)
for(i in 1:nsimul)
{
          print(plot_scalelocation(simulatedregs[[i]],NA))
}
dev.off()

pdf("simulated-qqnormal.pdf",width=14.23,height=6.77)
for(i in 1:nsimul)
{
          print(plot_qqnormal(simulatedregs[[i]]))
}
dev.off()
```