Running Linear and Generalized Additive Models

Generating data

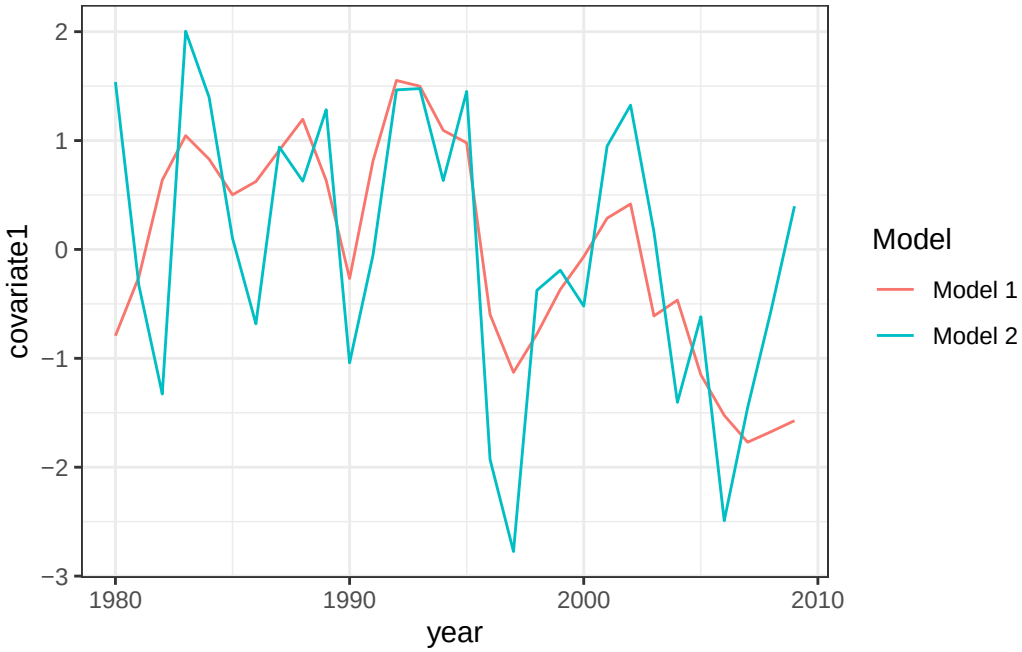
There are two ways we can fit these models:

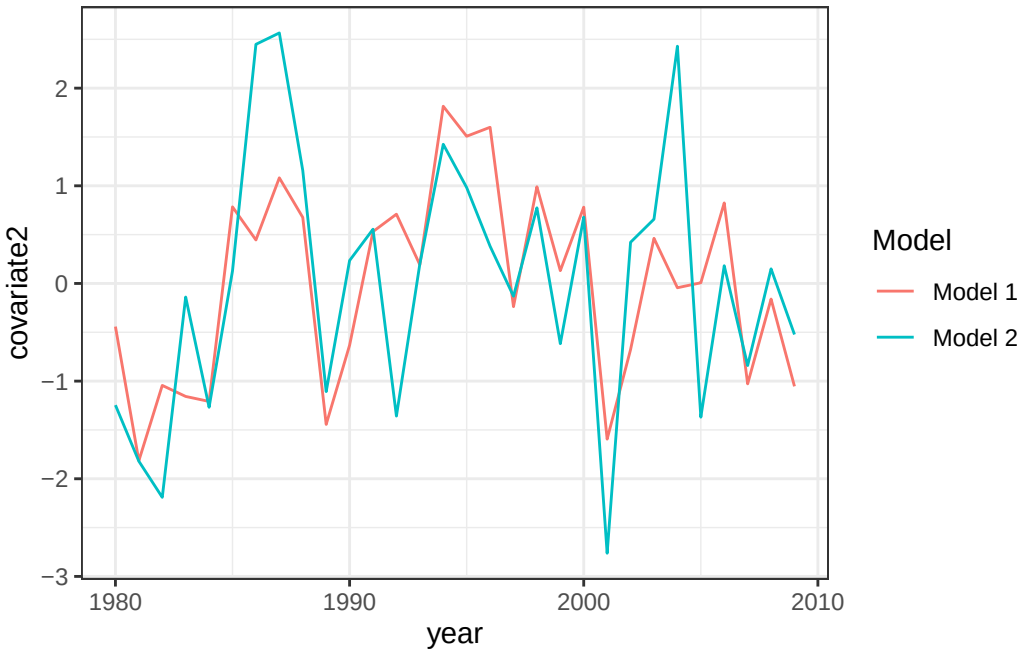
1) we can fit 2 separate models one for ROMs and one for GLORYs and compare r-squared and 2) we can fit 1 model and use a interaction term or model selection process. Doing #1 is easier and faster while #2 is better suited for a model selection technique.

To start, lets simulate some example data. Simulating data is a really great tool for testing models because you can simulate it with known or true relationships (based on values you assign).

```
set.seed(99)
year<- rep(seq(1980, 2009),2) # a variable for year
modelnames<- rep(c("Model 1", "Model 2"), each=30) # a variable for models,
#equivalent to ROMS/GLORYs

cov1mod1<- arima.sim(list(order = c(1,0,0), ar = 0.7), n = 30)%>% #random time series
  scale() # standardize it!
cov1mod2 <- cov1mod1+rnorm(n= 30, mean = 1, sd=0.3)%>%
  scale()#second time series that
#covaries with the first

set.seed(150)
cov2mod1<- arima.sim(list(order = c(1,0,0), ar = 0.5), n = 30)%>% #random time series
  scale() # standardize it!
cov2mod2 <- cov2mod1+rnorm(n= 30, mean =1, sd=1)%>%
  scale()
  #second time series that
#covaries with the first
```

```
dataset <- data.frame(year=year,covariate1=c(cov1mod1,cov1mod2),covariate2=c(cov2mod1,cov2mo
```





The two oceanographic models seem to track each other fairly well, BUT we want to be able

to say how well they track eachother through time. We can do this a few ways, covariance and Pearson's correlations are a good place to start!

```
#this tells us the covariance
cov1<- cov(dataset%>%filter(Model=="Model 1")%>%select(covariate1),
    dataset%>%filter(Model=="Model 2")%>%select(covariate1))

cov2<-cov(dataset%>%filter(Model=="Model 1")%>%select(covariate2),
    dataset%>%filter(Model=="Model 2")%>%select(covariate2))

#this tells us the correlation coefficient
cor1<- cor(dataset%>%filter(Model=="Model 1")%>%select(covariate1),
    dataset%>%filter(Model=="Model 2")%>%select(covariate1))

cor2 <-cor(dataset%>%filter(Model=="Model 1")%>%select(covariate2),
    dataset%>%filter(Model=="Model 2")%>%select(covariate2))

#lets store these in a dataframe so we have them
data.frame(covariate=c("covariate 1", "covarate 2"),
           covariance = c(cov1, cov2), correlation = c(cor1, cor2))
```

```
    covariate covariance correlation
1 covariate 1  0.8141265   0.6380151
2  covarate 2  0.8680522   0.6588066
```

These look to correlate and covary with eachother okay but not exceptionally well given they should be showing the same thing!

Note *for your standardized and unstandardized time series analysis you will want to include these two stats. You could make a table OR you could add these values to your figure either using annotate_figure or by putting them in the title*

Now lets simulate the recruitment data, lets pretend recruitment has a linear relationship with covariate 2 and a dome shaped relationship with covariate 2. Let pretend model 1 is a better predictor of Y rec

```
linear1 <- 4
linear2 <- 0.6
quad2<- 5
intercept <- 1
error<- rnorm(30, 0.1,0.5)

#this is just a simple qudratic equation with 2 covraiates plus random error
```

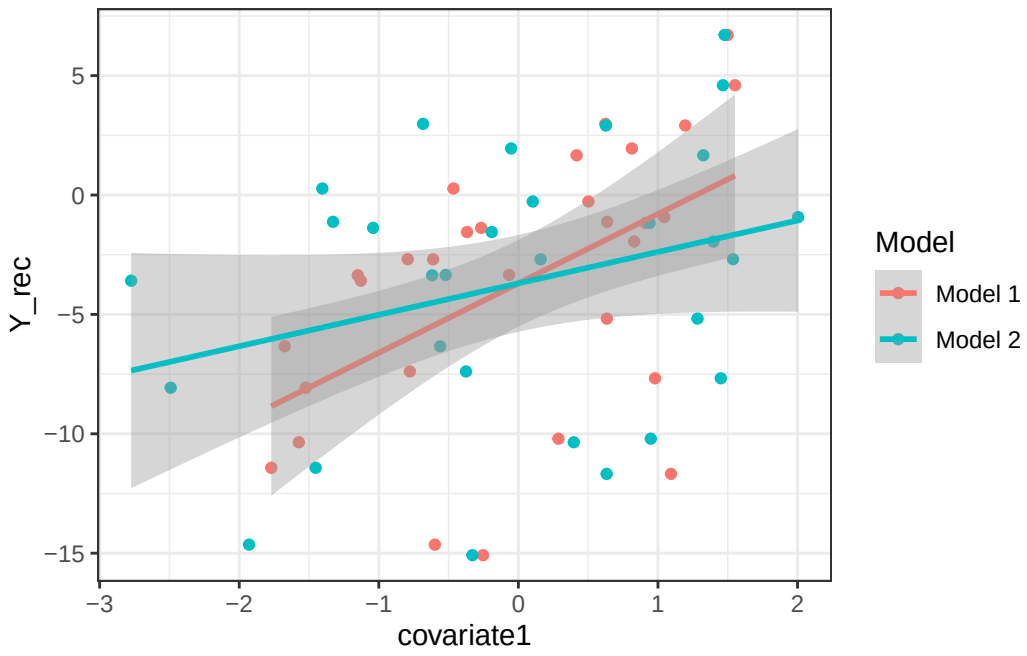
```r
Y_rec <- linear1*cov1mod1-linear2*cov2mod1-quad2*cov2mod1^2+error+intercept

data_full<-dataset%>%
  mutate(Y_rec=rep(Y_rec, 2))
```

Visually examining the relationships

The first way we can think about an oceanographic condition as a predictor/driver of recruitment is by looking at it visually. Lets look at the relationships between covariates/predictors and recruitment.

```r
ggplot(data=data_full, aes(x=covariate1, y=Y_rec, group=Model, col=Model))+
  geom_point()+
  geom_smooth(method='lm')+
  theme_bw()
```

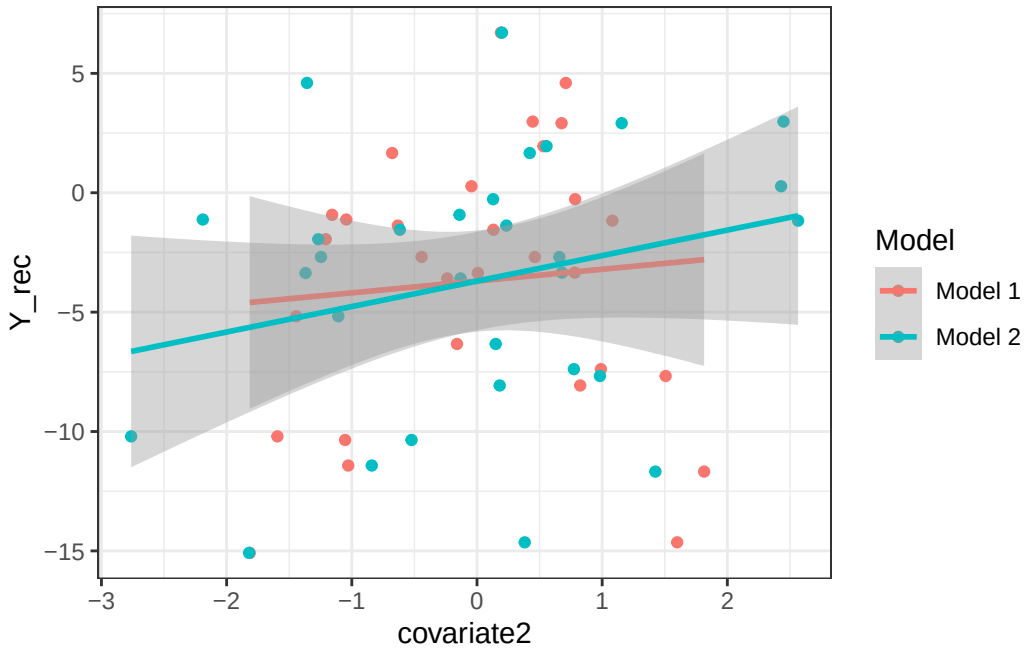
```
`geom_smooth()` using formula = 'y ~ x'
```



It looks pretty good! The covariate data look fairly linearly related, but it is impossible to tell which model does a better job just by looking.

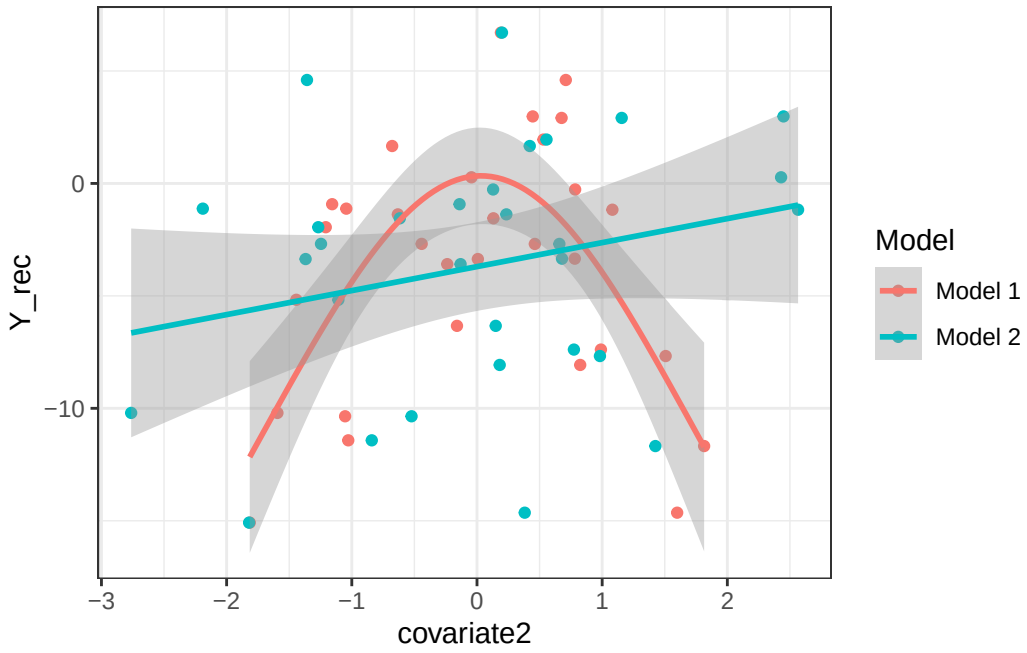
```
ggplot(data=data_full, aes(x=covariate2, y=Y_rec, group=Model, col=Model))+
  geom_point()+
  geom_smooth(method='lm')+
  theme_bw()
```

```
`geom_smooth()` using formula = 'y ~ x'
```



Hmm. this looks like a mess. These do not look like great linear relationships. What if we try looking at a more flexible model that can handle more relationship shapes! **This formulation of the geom_smooth() call is what I reccommend you use on your data** because it will fit both a line and a smoothed relationship.

```
ggplot(data=data_full, aes(x=covariate2, y=Y_rec, group=Model, col=Model))+
  geom_point()+
  geom_smooth(method = "gam", formula = y ~ s(x, k = 3))+
  theme_bw()
```



Much better! But it appears model 1 and 2 explain the data very differently and have very different functional relationships. For these plots we were just using the GAM and LM function built into ggplot. That is a great first pass that would be useful with the data you are using! BUT we also want actual information about the fits and the parameters that we can only get from actually fitting the model…so lets try fitting a some linear and generalized additive models first.

GAMs

Lets start with GAMs these are what were used for the Yellowtail stock assessment so this is a more useful place to start.

GAM stand for a generalized additive model. It is useful and used frequently in ecology because it allows for more flexible "functional relationships" aka the relationship between your driver and predictor, than linear models. For linear models you need to use quadratic or cubic terms to fit relationships that do not follow a line, and as a result they are "rigid". Rather than fitting a linear relationship, GAMs fit a "smoothed" relationship which is a fancy way of saying they use a function or equation to fit that relationship.

Lets fit separate GAMs for each model and covariate. I am going to just copy and paste the code here, rather than write it in a function so it is easier to follow. There are three metrics of this output we are interested in: 1) the r-squared, 2) the deviance explained, and 3) the significance or p-value of the smoothed term. Based on how we simulated the data above,

Model 1 should better explain the data than Model 2 so we should see that reflected in our results!

```
data_full_long <- pivot_longer(data_full, -c(Y_rec, Model, year), names_to = "covariatename"

gamoutput<-function(modelname, covariate){
gamfit<- gam(Y_rec~s(Value,k=3), data=data_full_long%>%filter(Model == modelname, covariatena

summary <- summary(gamfit)
predictions <- data.frame(prediction=predict(gamfit, se.fit=TRUE),year=unique(data_full$year)
  mutate(covariate=covariate, Model = modelname)

#pulling out the values we need and adding to a dataframe
results <- data.frame(covariate=covariate, Model = modelname, pvalue = summary$s.table[, "p-v

smooth_data <- smooth_estimates(gamfit)%>%  # Or specify the smooth term you want to plot
  add_constant(model_constant(gamfit)) %>% # Add the intercept
  add_confint()%>% # Add the credible interval
 # pivot_longer(c(covariate),names_to = "Smooth",values_to ="Value")%>%
  select(-c(.by))%>%
  mutate(covariate=covariate, Model = modelname)



output<-list(summary, predictions, results,smooth_data)
return(output)
  }
gamM1C1<-gamoutput("Model 1", "covariate1")
gamM2C1<-gamoutput("Model 2", "covariate1")
gamM1C2<-gamoutput("Model 1", "covariate2")
gamM2C2<-gamoutput("Model 2", "covariate2")

predictions_gam<- data.frame(gamM1C1[[2]])%>%
  bind_rows(data.frame(gamM2C1[[2]]))%>%
  bind_rows(data.frame(gamM1C2[[2]]))%>%
  bind_rows(data.frame(gamM2C2[[2]]))

results_gam<- data.frame(gamM1C1[[3]])%>%
  bind_rows(data.frame(gamM2C1[[3]]))%>%
  bind_rows(data.frame(gamM1C2[[3]]))%>%
  bind_rows(data.frame(gamM2C2[[3]]))

smooths_gam<- data.frame(gamM1C1[[4]])%>%
```

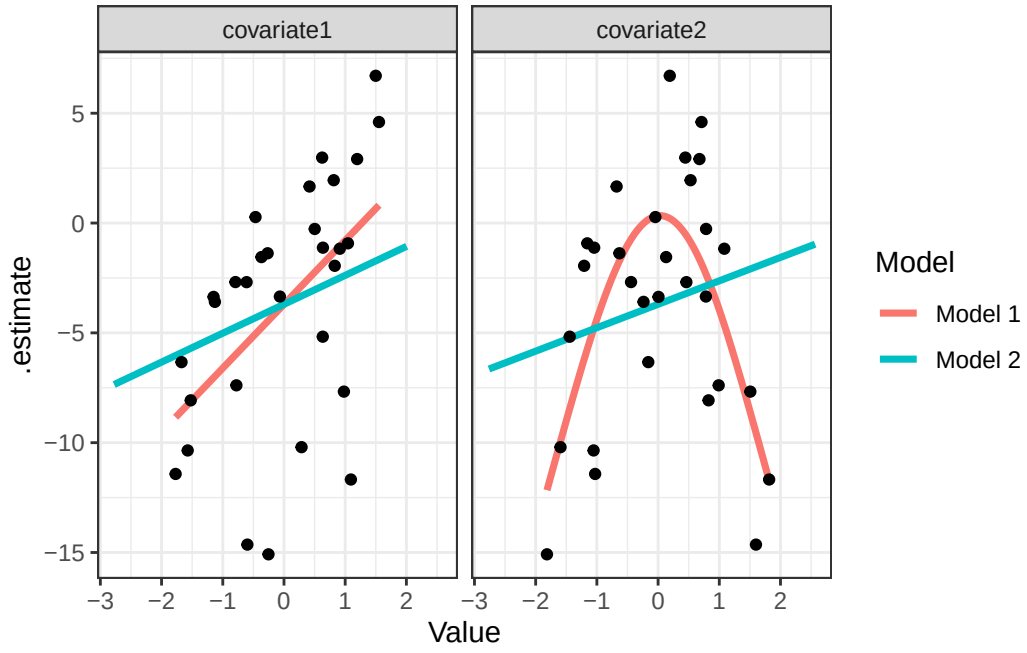
```
  bind_rows(data.frame(gamM2C1[[4]]))%>%
  bind_rows(data.frame(gamM1C2[[4]]))%>%
  bind_rows(data.frame(gamM2C2[[4]]))
```

```
   covariate   Model       pvalue DevianceExp        rsq
1 covariate1 Model 1 0.0032467197  0.27011770 0.27011770
2 covariate1 Model 2 0.1066880511  0.09028388 0.09028388
3 covariate2 Model 1 0.0001183199  0.49764253 0.49764253
4 covariate2 Model 2 0.1801757818  0.06322172 0.06322172
```

Based on these results, we find that Model 1 has a significant smooth for both covariates, it fits the data better than Model 2 (r-squared) AND it explains more deviance. Overall, Model 1 is a better predictor of recruitment than Model 2, despite the fact that both models covary and correlate quite well through time.

The last thing we want to do to compare these relationships is to look at the functional relationships for each model. This is quite similar to what we did above, BUT we want to do it for the actual model we fit rather than just using what is in ggplot automatically. This is a little annoying to do for gams, but the function above will give you the output you need!

```
Joining with `by = join_by(Value, Model)`
```



Conclusion: The conclusion that can be drawn from this example is that the difference between Model 1 and Model 2 are substantial enough to impact how we estimate a index of recruitment, including the functional relationship between and oceanographic condition and recrutiment.

Diandre' START HERE :)

I wrote a function for you to fit the GAMs so you basically just need to call the function with the covariate and model name. You can add the recruitment deviation data to your "long" data table, and then use this function to fit the gam called 'gamoutput'. The output will be returned to you as a list with 4 components: 1) the first part is the GAM summary 2) the second part are the model predictions of the gam 3) the results of the gam which includes the important values of the summary, the r-squared value, the deviance explained, and the p-value of the smoother (aka is the covariate significant) 4) the smoothed terms that you can use to plot the relationships

Here is the chunk with the function you can copy into your code!

```
data_full_long <- pivot_longer(data_full, -c(Y_rec, Model, year), names_to = "covariatename"

#the function starts here
gamoutput<-function(modelname, covariate, data){

#this is the equation of the gam that filters the for a model (GLORYs or MOM6)
# and filters for a covariate (MLDpjuv, Tcop etc. etc.)
gamfit<- gam(Y_rec~s(Value,k=3), data=data%>%filter(Model == modelname, covariatename==covari


summary <- summary(gamfit) #this pulls out the summary of the model

#this pulls out the model preditions and the SEs of the predictions and stores
#them into a dataframe
predictions <- data.frame(prediction=predict(gamfit, se.fit=TRUE),year=unique(data_full$year)
  mutate(covariate=covariate, Model = modelname)

#pulling out the values we need from the summary and adding to a dataframe
#that we can easily make a table out of
results <- data.frame(covariate=covariate, Model = modelname, pvalue = summary$s.table[, "p-v

#pulling out the shape of the relationship so that we can plot it!
smooth_data <- smooth_estimates(gamfit)%>%  # Or specify the smooth term you want to plot
  add_constant(model_constant(gamfit)) %>% # Add the intercept
```

```
  add_confint()%>% # Add the credible interval
 # pivot_longer(c(covariate),names_to = "Smooth",values_to ="Value")%>%
  select(-c(.by))%>%
  mutate(covariate=covariate, Model = modelname)

#putting it together as a list so we can get a single output with all the great
#info that we need!

output<-list(summary, predictions, results,smooth_data)
return(output)
}
```

Okay, so starting here is where you want to modify the code. I did not take the time to write this super efficiently here...lots of copy and pasteing. I will show you how to do it better in the next chunk!

To start, I call the gamoutput function written above with each model name, GLORYs or MOM6 and each oceanographic condition name.

```
#this runs the function for both ROMS and GLORYs for each covariate, you will
#want to change the values to match your data. Output list will be stored
# as a variable.

gamM1C1<-gamoutput("Model 1", "covariate1", data_full_long)
gamM2C1<-gamoutput("Model 2", "covariate1", data_full_long)
gamM1C2<-gamoutput("Model 1", "covariate2", data_full_long)
gamM2C2<-gamoutput("Model 2", "covariate2", data_full_long)


unique(data_full_long$Model)
```

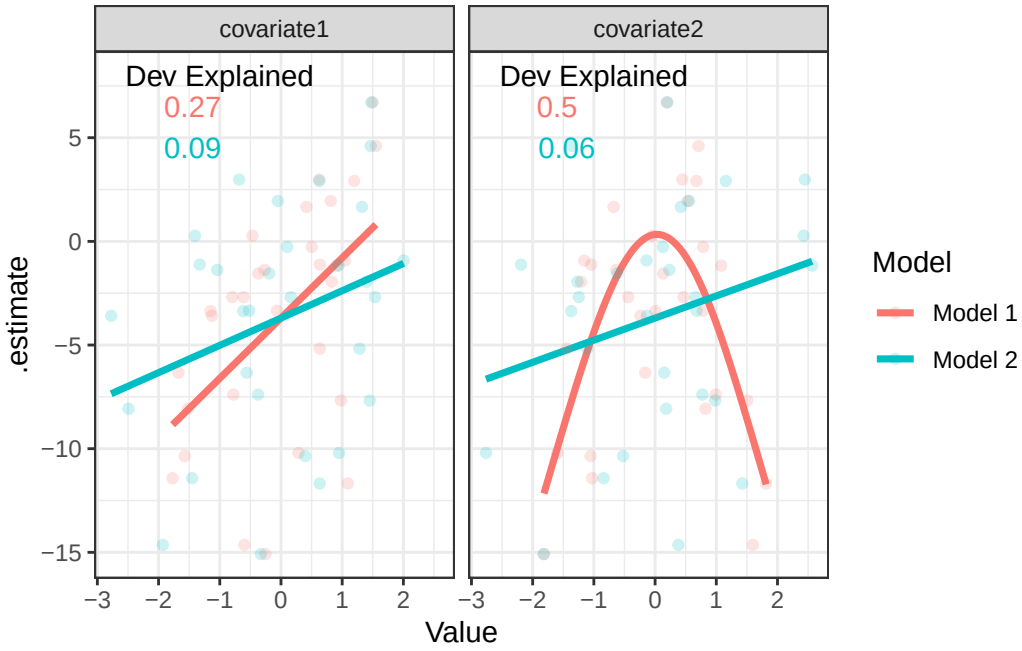
```
[1] "Model 1" "Model 2"
```

```
#now we can pull out each output we want from the list and combine it with the
# other runs of the function into a single dataframe

predictions_gam<- data.frame(gamM1C1[[2]])%>%
  bind_rows(data.frame(gamM2C1[[2]]))%>%
  bind_rows(data.frame(gamM1C2[[2]]))%>%
  bind_rows(data.frame(gamM2C2[[2]]))
```

```r
results_gam<- data.frame(gamM1C1[[3]])%>%
  bind_rows(data.frame(gamM2C1[[3]]))%>%
  bind_rows(data.frame(gamM1C2[[3]]))%>%
  bind_rows(data.frame(gamM2C2[[3]]))

smooths_gam<- data.frame(gamM1C1[[4]])%>%
  bind_rows(data.frame(gamM2C1[[4]]))%>%
  bind_rows(data.frame(gamM1C2[[4]]))%>%
  bind_rows(data.frame(gamM2C2[[4]]))
```

Now we have all the output we want stored in dataframes. The next thing to do is to make plots from that output. From the code above you will have a "smooth_full" obect which you can now use to make your plots once you join it with you dataset. For you, this plot will have 9 panels like your time series plots!

```
Joining with `by = join_by(Value, Model)`
Joining with `by = join_by(covariate, Model)`
```



Here is another option using a loop rather than copy and pasting! It will loop over each model, then each covariate for each model

```r
#here is a more efficient way to run the function through a loop to
# generate the same output from the copy paste above!
#this creates the same objects that you can run through the plot code


Modelnames <- unique(data_full_long$Model) #change this to your dataset
covariatenames <- unique(data_full_long$covariatename) #change this to your dataset
predictions_gam2<- data.frame()
results_gam2<- data.frame()
smooths_gam2<- data.frame()

for(i in 1:length(covariatenames)){
  for(j in 1:length(Modelnames)){
    gamout<-gamoutput(Modelnames[j], covariatenames[j], data_full_long)
    predictions_gam2<-rbind(predictions_gam2, gamout[[2]])
    results_gam2<-rbind(results_gam2, gamout[[3]])
    smooths_gam2<-rbind(smooths_gam2, gamout[[4]])
  }
  print(i)
}
```

```
[1] 1
[1] 2
```