swnsmodelr R Package Manual

Dana Melamed

June 7, 2018

## Introduction

The swnsmodelr package was designed for users to easily model SWNS weather station data with rasters and models of their chosing. The package contains functions and weather station data, and input rasters are contained within the package’s R project. This document follows a workflow of preparing data, generating models, and generating outputs. The objects generated at the various steps in the example are stored in swnsmodelr.

## Exract Rasters to Weather Station Data

The first step in modelling weather station data with rasters is to extract raster values to corresponding weather station records. For many rasters, (ex: elevation, proximity to coast, aspect) the value will be same every day for a station. Other rasters (ex: solar radiation) change on a temporal basis, and will have a unique value for each station every day. The two type of rasters, constant, and temporally changing, will be treated differently in the value extract process.

### Weather Stations Data Frame

The data frame, swns\_stations\_df, stored within swnsmodelr, contains SWNS daily temperature data from 2012 - 2017. The data frame has columns for: station I.D. (stationid), date (date\_time), minimum temperature (temp\_min), maximum temperature (temp\_max), and mean temperature (temp\_mean). One row of data represents one daily record for one station. Use str(), to see the column names and variables types.

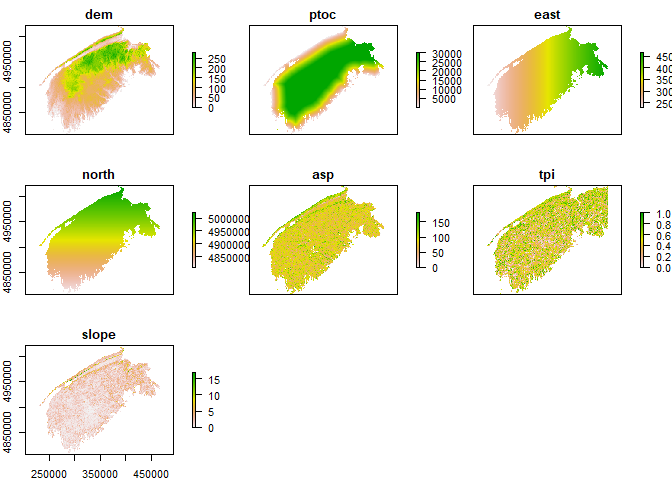
str(swns\_stations\_df)

## 'data.frame': 188163 obs. of 7 variables:  
## $ stationid: chr "27141" "27141" "27141" "27141" ...  
## $ date\_time: Date, format: "2012-01-01" "2012-01-02" ...  
## $ temp\_min : num -0.8 1 -9.8 -12.2 -8 -10.5 -5.4 -2.4 -9.6 -8.1 ...  
## $ temp\_mean: num 2.9 6.4 -3.9 -10 -3.9 -7.8 -0.5 1.1 -5.9 -1.7 ...  
## $ temp\_max : num 6.5 11.7 2.1 -7.8 0.3 -5 4.4 4.5 -2.2 4.7 ...  
## $ EASTING : num 383226 383226 383226 383226 383226 ...  
## $ NORTHING : num 4991426 4991426 4991426 4991426 4991426 ...

### Constant Rasters

The constant rasters should be stored as a list of raster objects. The code below is an example of how to create a list of raster objects from rasters available in the package project. The rasters are from the 200 subdirectory within the Rasters folder. In this subdirectory, rasters have a spatial resolution of 200m.

# Names of rasters in Rasters directory (everything before extension)  
rasters\_names\_list <- list("dem", # elevation  
 "ptoc", # proxmity to coast  
 "east", # easting  
 "north",# northing  
 "asp", # aspect  
 "tpi", # topographic position index  
 "slope") # slope  
# Make into rasters objects list  
rasters\_list <- lapply(FUN = raster,  
 X = paste0("E:\\Packages\\swnsmodelr\\Rasters\\200\\",rasters\_names\_list, ".tif"))  
  
# Plot brick of rasters  
rasters\_list %>% brick() %>% plot()



The constant raster values are added to the swns\_stations\_df table with extract\_constant\_rasters\_values(). A new frame will be made called swns\_stations\_df\_200, which is the original table plus constant raster values at 200m resolution

swns\_stations\_df\_200 <- extract\_constant\_raster\_values(swns\_stations\_df, rasters\_list)  
str(swns\_stations\_df\_200)

## 'data.frame': 188163 obs. of 14 variables:  
## $ stationid: chr "27141" "27141" "27141" "27141" ...  
## $ date\_time: Date, format: "2012-01-01" "2012-01-02" ...  
## $ temp\_min : num -0.8 1 -9.8 -12.2 -8 -10.5 -5.4 -2.4 -9.6 -8.1 ...  
## $ temp\_mean: num 2.9 6.4 -3.9 -10 -3.9 -7.8 -0.5 1.1 -5.9 -1.7 ...  
## $ temp\_max : num 6.5 11.7 2.1 -7.8 0.3 -5 4.4 4.5 -2.2 4.7 ...  
## $ EASTING : num 383226 383226 383226 383226 383226 ...  
## $ NORTHING : num 4991426 4991426 4991426 4991426 4991426 ...  
## $ dem : num 59.2 59.2 59.2 59.2 59.2 ...  
## $ ptoc : num 17782 17782 17782 17782 17782 ...  
## $ east : num 383177 383177 383177 383177 383177 ...  
## $ north : num 4991432 4991432 4991432 4991432 4991432 ...  
## $ asp : num 131 131 131 131 131 ...  
## $ tpi : num 0.796 0.796 0.796 0.796 0.796 ...  
## $ slope : num 1.83 1.83 1.83 1.83 1.83 ...

### Temporally Changing Rasters

In this modelling procedure, a solar radiation raster for each day is used an input into the model. The large quantity of rasters makes it problematic to store rasters as a list of raster objects, as with the constant rasters. To overcome this issue, there are two steps to adding temporally changing rasters to swns\_stations\_df. In the first step, the file path to each raster is stored as a row in a new dataframe. The date is then parsed from the file path name, to a new column of corresponding dates. This way, rasters can be selected and converted to rasters objects based on dates. The data frame, referred to as a ‘temporal rasters data frame’, is made with make\_temporal\_rasters\_df(). The user must define the “date characters” or date\_chars argrument, which is the index of characters that contain the date in each file path. The date format (date\_format agrument) must also be defined (see ?as.Date). The user also defines a start and end date of rasters to include, and the extension of the rasters. Using the file “Sum\_Irradiance\_2012\_1.tif” as an example: the date is formed with the characters 16 to -5, and the format is “%Y\_%j”. A sample use of the function for the solar radiation rasters is shown below:

solar\_irradiance\_rasters\_df <- make\_temporal\_raster\_df(  
 in\_folder = "E:\\GOES\_200m\_2",  
 start\_date = ymd('2012-01-01'),  
 end\_date = ymd('2012-12-31'),  
 date\_chars = c(16,-5),  
 date\_format = "%Y\_%j",  
 extension = ".tif")  
  
head(solar\_irradiance\_rasters\_df)

## path\_field date\_time  
## 1 E:\\GOES\_200m\_2\\Sum\_Irradiance\_2012\_1.tif 2012-01-01  
## 2 E:\\GOES\_200m\_2\\Sum\_Irradiance\_2012\_10.tif 2012-01-10  
## 3 E:\\GOES\_200m\_2\\Sum\_Irradiance\_2012\_100.tif 2012-04-09  
## 4 E:\\GOES\_200m\_2\\Sum\_Irradiance\_2012\_101.tif 2012-04-10  
## 5 E:\\GOES\_200m\_2\\Sum\_Irradiance\_2012\_102.tif 2012-04-11  
## 6 E:\\GOES\_200m\_2\\Sum\_Irradiance\_2012\_103.tif 2012-04-12

Finally, to add the values of each temporal raster to the weather station data, use extract\_temporal\_raster\_values(). This may take a while if there are many temporal rasters. The function add a column for the set of temporally changing rasters. The argument col\_name is used to specify the name of the column. By setting verbose to true, the function will print a completion report for every raster that has been succesfully extracted to the weather stations data frame. The output data frame was named swns\_stations\_df\_200, because it has the raster values extracted from 200m resolution.

swns\_stations\_df\_200 <- extract\_temporal\_raster\_values(temporal\_rasters\_df = solar\_irradiance\_rasters\_df,  
 temperatures\_df = swns\_stations\_df\_200,  
 col\_name = "sum\_irradiance",  
 verbose = FALSE)

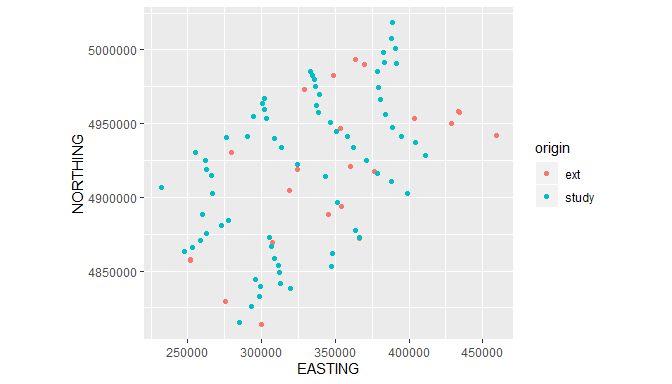
head(swns\_stations\_df\_200)

## stationid date\_time temp\_min temp\_mean temp\_max EASTING NORTHING dem  
## 1 27141 2012-01-01 -0.8 2.9 6.5 383226.3 4991426 59.225  
## 2 27141 2012-01-02 1.0 6.4 11.7 383226.3 4991426 59.225  
## 3 27141 2012-01-03 -9.8 -3.9 2.1 383226.3 4991426 59.225  
## 4 27141 2012-01-04 -12.2 -10.0 -7.8 383226.3 4991426 59.225  
## 5 27141 2012-01-05 -8.0 -3.9 0.3 383226.3 4991426 59.225  
## 6 27141 2012-01-06 -10.5 -7.8 -5.0 383226.3 4991426 59.225  
## ptoc east north asp tpi slope sum\_irradiance  
## 1 17782.27 383177.3 4991432 130.8961 0.7963742 1.832468 533.4339  
## 2 17782.27 383177.3 4991432 130.8961 0.7963742 1.832468 441.0299  
## 3 17782.27 383177.3 4991432 130.8961 0.7963742 1.832468 990.5778  
## 4 17782.27 383177.3 4991432 130.8961 0.7963742 1.832468 909.8625  
## 5 17782.27 383177.3 4991432 130.8961 0.7963742 1.832468 388.7767  
## 6 17782.27 383177.3 4991432 130.8961 0.7963742 1.832468 1445.6809

## Separate Stations into Modelling and Validating Stations

There are weather stations from AGRG (study stations), ECAN and DNR (external stations) in the data frame. Ideally, the study stations would be used to model the data, and the external stations to validate the models. To be able to differeniate at any time between study (AGRG) and external (ECAN and DNR) stations, the station I.D.’s were sorted into two lists: study\_stations\_list and ext\_stations\_list. In the code below, the lists were used to separate the stations, and they were plotted to show their locations.

ggplot(data = swns\_stations\_df %>%  
 filter(!duplicated(stationid)) %>%  
 dplyr::mutate(origin = if\_else(stationid %in% study\_stations\_list,  
 "study",  
 "ext"))) +  
 geom\_point(aes(x = EASTING, y = NORTHING, colour = origin)) +  
 coord\_fixed()



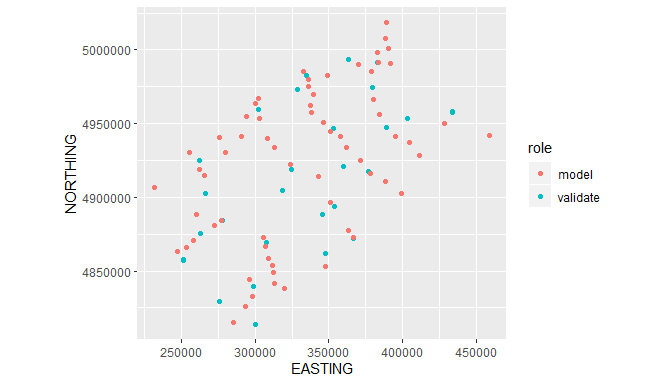
The study stations alone do not provide sufficient coverage in the eastern most portion surrounding the Halifax area, and the Annapolis valley. There are external stations in those areas which were included in the model stations along with the study stations. The external stations that were included as model stations were ‘6354’ in Greenwood, ‘6456’ north is St. Margaret’s Bay, ‘47187’ in Halifax, ‘DN025’ south of Digby and ‘6497’ near Berwick. A number of study stations were used in validating the models rather than generating them, these stations were: “YA4”,“S160”,“AN3”,“CL4”,“KE5”,“WE5”,“WE2”,“LI2”,“CH4”. These stations are located in areas represented by model stations but not so close that their modelling values were essentially the same.

The following code was used to separate the stations, and can be easily altered to exclude or include stations in either category:

# Choose modelling stations  
model\_stations\_df <- swns\_stations\_df\_200 %>%  
 # 1. Include all study stations  
 filter(stationid %in% study\_stations\_list |  
 # 2. Included these ext stations  
 stationid %in% c("6354","47187","6456","DNR025","6497")) %>%   
 # 3. Remove these study stations  
 filter((stationid %in% c("YA4","S160","AN3","CL4","KE5","WE5","WE2","LI2","CH4")) == FALSE)  
  
# Choose validation stations  
val\_stations\_df <- swns\_stations\_df\_200 %>%   
 # 1. Include all ext stations  
 filter(stationid %in% ext\_stations\_list |   
 # 2. Add these study stations  
 stationid %in% c("YA4","S160","AN3","CL4","KE5","WE5","WE2","LI2","CH4")) %>%  
 # 3. Remove these ext stations  
 filter(!(stationid %in% c("6354", "47187", "6456","27141", "DLG006", "DNR024",  
 "6501", "DNR025", "DNR003", "6497")   
 )  
 )

This is the resulting modelling and validation stations:

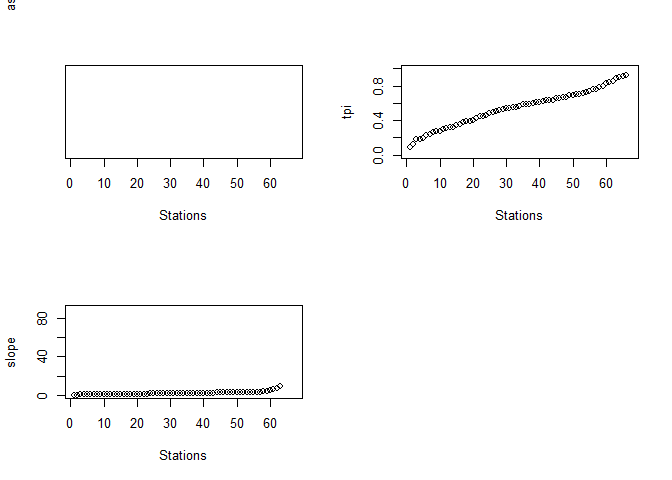
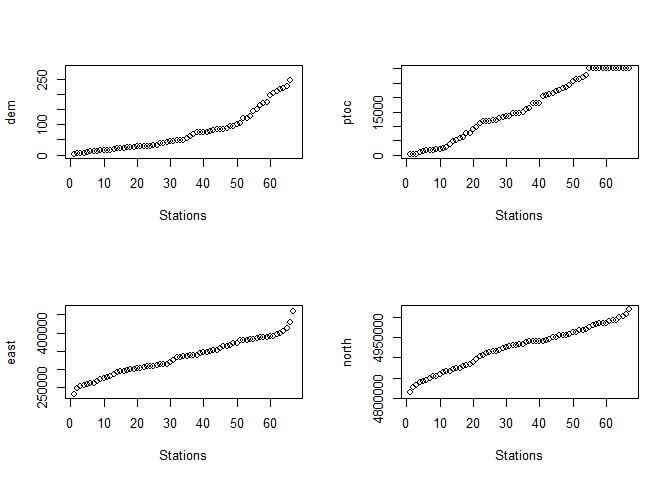
ggplot(data = swns\_stations\_df %>%  
 filter(!duplicated(stationid)) %>%  
 dplyr::mutate(role = if\_else(stationid %in% unique(model\_stations\_df$stationid),  
 "model",  
 "validate"))) +  
 geom\_point(aes(x = EASTING, y = NORTHING, colour = role)) +  
 coord\_fixed()



## Examine Raster Variable Relationship with Weather Station Temperature

### Weather Station Coverage of Raster Variability

model\_stations\_sp <- model\_stations\_df %>%   
 filter(!duplicated(stationid)) %>%  
 dplyr::select(stationid, EASTING, NORTHING)  
coordinates(model\_stations\_sp) = ~ EASTING + NORTHING  
  
par(mfrow = c(2,2))  
for(i in seq\_along(rasters\_list)){  
 raster\_values <- raster::extract(rasters\_list[[i]], model\_stations\_sp)   
 if(names(rasters\_list[[i]]) == "slope"){  
 plot(x = 1:(length(raster\_values)),  
 y = raster\_values[sort.list(raster\_values)],   
 ylab = names(rasters\_list[[i]]),  
 xlab = "Stations",  
 ylim = c(0,90))  
 }else{  
 plot(x = 1:(length(raster\_values)),  
 y = raster\_values[sort.list(raster\_values)],   
 ylab = names(rasters\_list[[i]]),  
 xlab = "Stations",  
 ylim = c(minValue(rasters\_list[[i]]), maxValue(rasters\_list[[i]])))  
}  
   
}



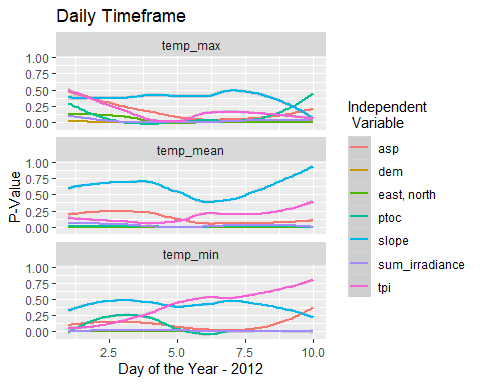
### Test Dependent Variable/Independent Variable Combinations at Different Timeframes

In the swns\_stations\_df\_200 data frame there are three dependent variables: daily minimum, maxmimum, and mean temperatures, and seven independent variables extracted from rasters. Functions were written to model dependent and independent variables one-on-one, to see the individual relationships. A function was written for the daily, weekly, and monthly timeframes: extract\_daily\_spvs(), extract\_weekly\_spvs, extract\_monthly\_spvs. The term “spvs” refers to “smooth term p-values”. The functions a data frame that is similar to the input data frame, with three added columns: ind\_var and dep\_var, which specify which columns in the row were used as independent and dependent variables, respectively; and pval, which is the p-value from the smooth term in the GAM.

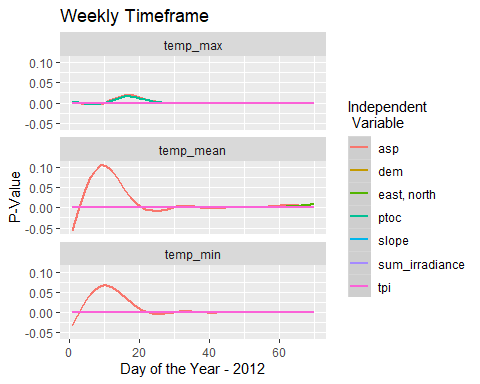
A sample of using the three functions, and plotting their results, are in the code examples below.

swns\_stations\_df\_200 <- add\_date\_columns(swns\_stations\_df\_200)  
daily\_spvs\_df <- extract\_daily\_spvs(swns\_stations\_df\_200, # data frame  
 c("temp\_min","temp\_max",  
 "temp\_mean"), # dependent variables  
 c("dem","asp","slope","tpi","ptoc",  
 "sum\_irradiance","east, north"), #independent variables  
 2012, # year  
 1:10) # days in the year (ex: 1:365)

ggplot(data = daily\_spvs\_df, aes(x = yday, y = pval)) +  
 geom\_smooth(aes(colour = ind\_var)) +  
 facet\_wrap(~dep\_var, ncol = 1) +  
 labs(x = "Day of the Year - 2012", y = "P-Value",  
 title = "Daily Timeframe",  
 colour = "Independent \n Variable")



swns\_stations\_df\_200 <- add\_date\_columns(swns\_stations\_df\_200)  
daily\_spvs\_df <- extract\_weekly\_spvs(swns\_stations\_df\_200, # data frame  
 c("temp\_min","temp\_max",  
 "temp\_mean"), # dependent variables  
 c("dem","asp","slope","tpi","ptoc",  
 "sum\_irradiance","east, north"), #independent variables  
 2012, # year  
 1:10) # weeks in the year (ex: 1:52)  
ggplot(data = daily\_spvs\_df, aes(x = yday, y = pval)) +  
 geom\_smooth(aes(colour = ind\_var)) +  
 facet\_wrap(~dep\_var, ncol = 1) +  
 labs(x = "Day of the Year - 2012", y = "P-Value",  
 title = "Weekly Timeframe",  
 colour = "Independent \n Variable")



swns\_stations\_df\_200 <- add\_date\_columns(swns\_stations\_df\_200)  
daily\_spvs\_df <- extract\_monthly\_spvs(swns\_stations\_df\_200, # data frame  
 c("temp\_min","temp\_max",  
 "temp\_mean"), # dependent variables  
 c("dem","asp","tpi","ptoc",  
 "sum\_irradiance","east, north"), #independent variables  
 2012, # year  
 1) # months in the year (ex: 1:365)  
ggplot(data = daily\_spvs\_df, aes(x = yday, y = pval)) +  
 geom\_smooth(aes(colour = ind\_var)) +  
 facet\_wrap(~dep\_var, ncol = 1) +  
 labs(x = "Day of the Year - 2012", y = "P-Value",  
 title = "Monthly Timeframe",  
 colour = "Independent \n Variable")

## Generate and Validate Models

A set of functions in swnsmodelr generate GAMs based on a model station data, then apply the models to validation station data, and computes the error at each validation record. The output from the functions is the validation station data frame with columns for the error (resid), GCV score(gcv), and adjusted R square () added. There is a function written for the three timeframes, daily, weekly, and monthly, respectively: validate\_daily\_GAMs(), validate\_weekly\_GAMs(), validate\_monthly\_GAMs().

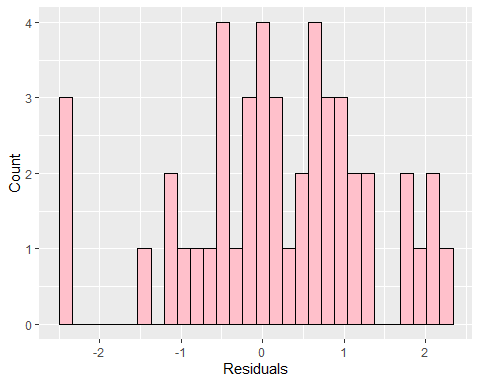
The functions allow the user to specify an alternative formula, with the argument alt\_formula. This feature was specifically implemented for the case that solar radiation data is missing for a particular date. The function will fail if the data is missing, and fall back on the alternative formula. If there is no suspiscion that a missing variable will cause the function to fail, this argument can be left blank.

daily\_val\_df <- validate\_daily\_GAMs(model\_stations\_df,  
 val\_stations\_df,  
 years = 2012,  
 days = 1:3,  
 formula = "temp\_mean ~ s(dem) + s(asp) + s(tpi)   
 + s(ptoc, k= 3) + s(east,north) + s(sum\_irradiance)",  
 alt\_formula ="temp\_mean ~ s(dem) + s(asp) + s(tpi)   
 + s(ptoc, k = 3) + s(east,north)")

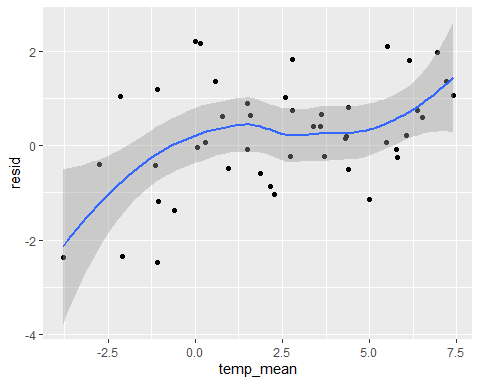
## Warning in if (is.na(daily\_res[[i]]$resid)) {: the condition has length > 1 and  
## only the first element will be used  
  
## Warning in if (is.na(daily\_res[[i]]$resid)) {: the condition has length > 1 and  
## only the first element will be used  
  
## Warning in if (is.na(daily\_res[[i]]$resid)) {: the condition has length > 1 and  
## only the first element will be used

Two ways of visualizing the errors are by histogram, and a scatterplot against the dependent variable.

ggplot(data = daily\_val\_df, aes(resid))+  
 labs(x = "Residuals",y = "Count") +  
 geom\_histogram(position = "dodge", colour = "black", fill = "pink",bins = 30)



ggplot(data = daily\_val\_df, aes(x = temp\_mean, y = resid)) +  
 geom\_point() +  
 geom\_smooth()



## Generating Output

### Overview

With the combination of a model stations data frame, model formula, and input rasters, functions in swnsmodelr generate different types of raster outputs. The functions daily\_model\_predict\_rasters(), and weekly\_model\_predict\_rasters() generates daily rasters of the dependent variable with the specified model applied in daily or weekly timeframes, respectively.

The prediction functions will fail if the rasters in the rasters\_list, and the rasters pointed to in the temporal\_rasters\_df, do not have the same extents, and number of rows and columns. The function raster::compareRaster() will be return TRUE for the rasters\_list if the properties are the same. If not, the function raster::resample() can be used to correct the issue.

The function generate\_mean\_rasters(), calculates mean rasters for sets of maximum and minimum rasters that correpsond by date. This function was written incase a user would rather model daily minimum and maximum temperatures before calculating daily mean temperature.

Finally, the function generate\_output\_gdd(), generates GDD rasters from daily mean temperature rasters. The user can specify the GDD base, and frequency of outputs (daily, weekly, monthly or yearly). There is also an agrument called growing\_season that when set to TRUE, only accumulates GDD for April to November.

### Predict Daily Mean Temperature Rasters

Continuing from the previous section, the code below demonstrates how to use daily\_model\_predict\_rasters().As with the model validation, a formula and an alternative formula (alt\_formula) can be specified. The alt\_formula is used if the formula causes gam() to fail. This would occur is a variable is a missing for the date being modelling, which is the case randomly with sum\_irradiance.

# Predict daily mean temperature  
daily\_model\_predict\_rasters(start\_date = ymd('2012-01-01'),  
 end\_date = ymd('2017-12-31'),  
 formula = "temp\_mean ~  
 s(east,north) +  
 s(dem) +  
 s(sum\_irradiance) +  
 s(ptoc, k= 3)",  
 alt\_formula = "temp\_mean ~  
 s(east,north) +  
 s(dem) +  
 s(ptoc, k= 3)",  
 temperatures\_df = model\_stations\_df,  
 input\_rasters = rasters\_list,  
 temporal\_rasters\_df = solar\_irradiance\_rasters\_df,  
 output\_folder = "F://daily\_models//mean\_temperature",  
 output\_ext = "tif",  
 verbose = TRUE)

### Calculate Growing Degree Day (GDD) Rasters

Rasters of GDD are calculated by raster algebra with the daily mean temperature rasters. The first step is to make a data frame with make\_temporal\_rasters\_df() that contains the file paths to the daily mean temperature rasters, and their corresponding date. The function output\_gdd\_rasters() applies the specified GDD base to the mean temperature rasters, accumulates them daily, and sends them to the output folder at the specified timeframe, output\_timeframe. The accumulation only occurs within a year, and can be set to only include April 1st - November 30th by setting growing\_season to TRUE. The code below demonstrates how to use the two functions to generate GDD rasters.

temp\_mean\_df <- make\_temporal\_raster\_df("F://daily\_models//mean\_temperature",  
 start\_date = ymd('2012-01-01'),  
 end\_date = ymd('2017-12-31'),  
 date\_chars = c(10,19),  
 date\_format = "%Y-%m-%d")  
  
generate\_gdd\_output(temp\_mean\_df,  
 gdd\_base = 5,  
 start\_date = start\_date,  
 end\_date = end\_date,  
 output\_time\_slice = "monthly",  
 growing\_season = TRUE,  
 output\_folder = "Z:\\Dana\\Weekly\\Monthly\_GDD\_200\_7",  
 plot\_gdd\_raster = TRUE)