Electric Car Modeling

packages:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library( scales)
library( fields)
```

```
## Loading required package: spam
```

```
## Spam version 2.9-1 (2022-08-07) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
```

```
## Loading required package: viridisLite
```

```
##
## Try help(fields) to get started.
```

#Hello this is the data exploration for electric car dataset

```
setwd("C:\\Users\\levil\\OneDrive\\Documents\\Data Science\\SpatialStats\\ElectricCarProject")
```

```
carData <- read.csv("cleaned_electric_car_data.csv")
```

going to have to do some more cleaning. Drew lied and said it was ready to go. It appears that it is EV car data from more than Washington.

```
LocationEVs <- carData[,c("Longitude", "Latitude")] %>%
  group_by(Longitude, Latitude) %>%
  count()

colnames(LocationEVs) <- c("lon","lat","evCount")
```

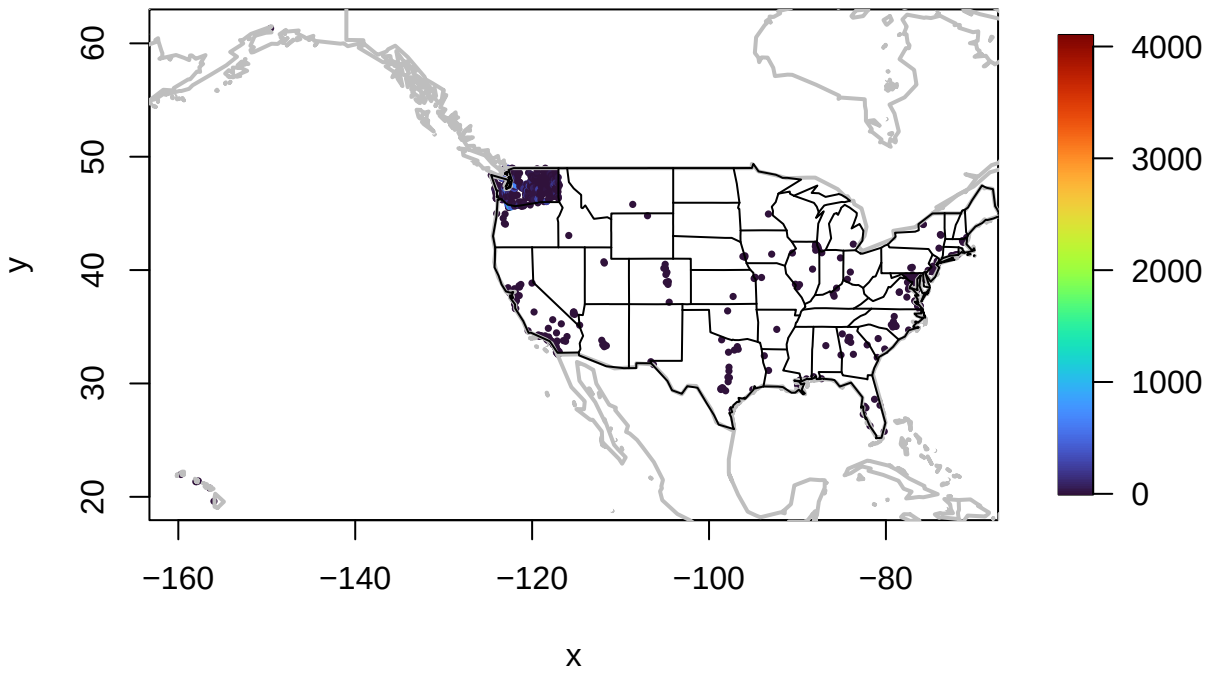
plot data to see if it worked correctly

```
LocationEVs[, 1:2]
```

```
## # A tibble: 835 x 2
## # Groups:   lon, lat [835]
##      lon   lat
##    <dbl> <dbl>
##  1 -160.  22.0
##  2 -158.  21.3
##  3 -158.  21.3
##  4 -158.  21.4
##  5 -156.  20.8
##  6 -156.  19.6
##  7 -150.  61.3
##  8 -125.  48.4
##  9 -124.  48.3
## 10 -124.  48.1
## # i 825 more rows
```

Just looking at this we need to remove the outlines for our data. As well I think it will be best to remove Hawaii and Alaska even though they have some counts they are not grouped with the original group which just causes problems. It probably will be best to just do Washington as well as that is where the majority of the data is. So I am going to create a dataset with just Washington and then plot that. We can then investigate further.

```
bubblePlot(LocationEVs[, 1:2],LocationEVs$evCount,  size=.5, col=turbo(256),highlight=FALSE)
world( add=TRUE, col="grey", lwd=2)
US(add=TRUE)
```

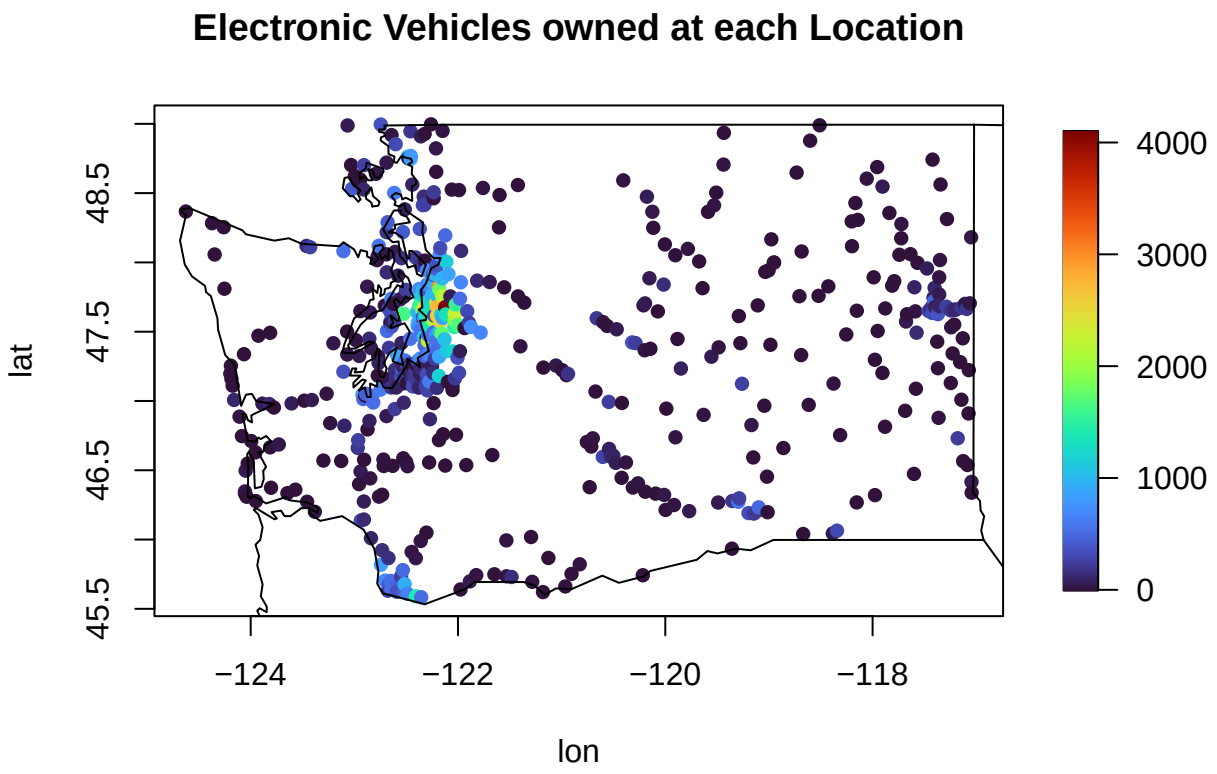


#Creating Just Washington Dataset

```
WashingtonEVs <- carData[,c("State", "Longitude", "Latitude")] %>%
  filter(State == "WA") %>%
  group_by(Longitude, Latitude) %>%
  count()

colnames(WashingtonEVs) <- c("lon","lat","evCount")
```

This looks much better and definitely will be more useful. When looking at this you can sort of see that there is really close to zero values in most of the rural areas. However, by citys such as Seattle or Portland the values are much much higher. This is probably due to multiple things. There is larger populations at cities so it work makes sense there is more cars and therefore more EV's. Even if there is a lower percentage of EV's there still will be more due to the increased population. As well with the increased infrastructure there is mostlikely more charging stations and therefore more viable to have an EV. If we want to account for the first part we could potentially model percentage of population that has an EV by taking our ev Values devided by total car ownership at the location.

```
bubblePlot(WashingtonEVs[,1:2],WashingtonEVs$evCount,  size=1, col=turbo(256), highlight=FALSE, ylab =
#world( add=TRUE, col="grey", lwd=2)
US(add=TRUE)
```



Now onto model fitting :)

Based on the size of this location as well as the number of datapoints across the state. I feel pretty good that we will just be able to fit spatial process. I don't think we will need to use fasttps or latus krig in order to get a fit and cut down on computation time. We should be able to use spatial process just fine.

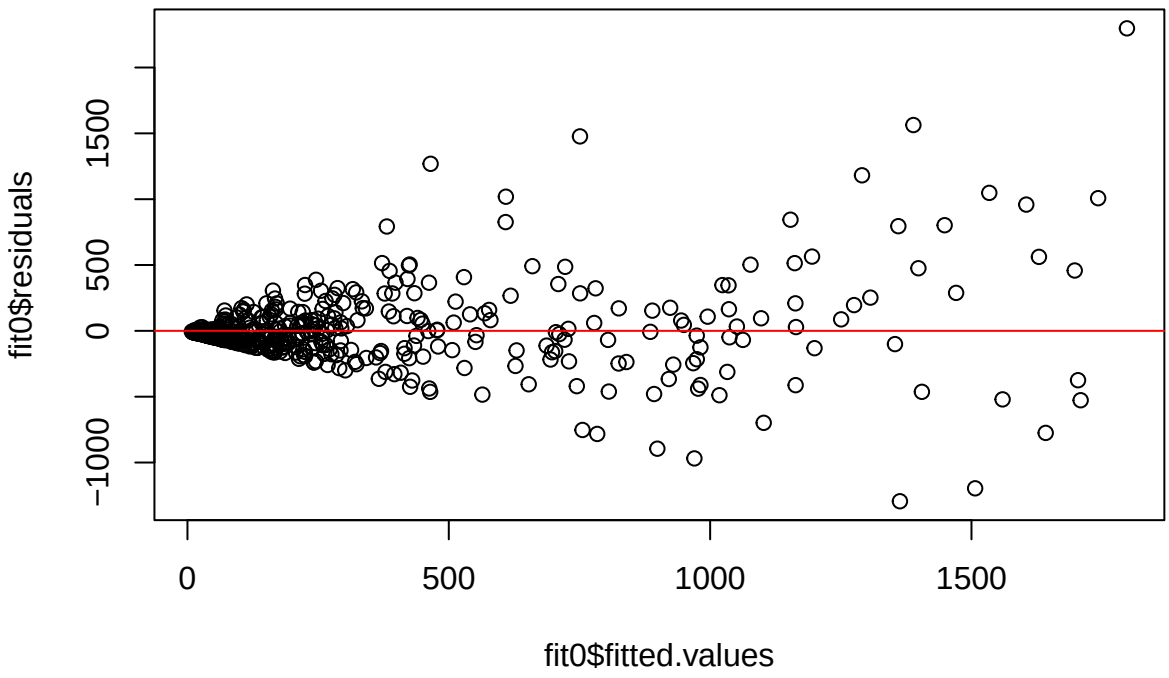
```
s <- WashingtonEVs[,1:2]
y <- WashingtonEVs[,3]
```

Just fitting a base spatial process to the model then can evaluate the results and move on from there.

```
fit0 <- spatialProcess(s, y)
```

Residuals vs fitted plot. The is definitely some concerns here. A pretty steady cornucopia with our data as there is a very close relationship between small residuals and fitted values being pretty close while larger values are much more varied. However, I feel as though we are going to struggle in completely overcoming this as most of the rural spots are around 0 or 1. So those are pretty easy to predict. It is having more trouble predicting cities which makes sense as they have more variance. However I do think we can do better. Potentially we transform our car counts with a log transformation and we can see what that does.

```
plot( fit0$fitted.values, fit0$residuals)
abline( h=0, col="red")
```

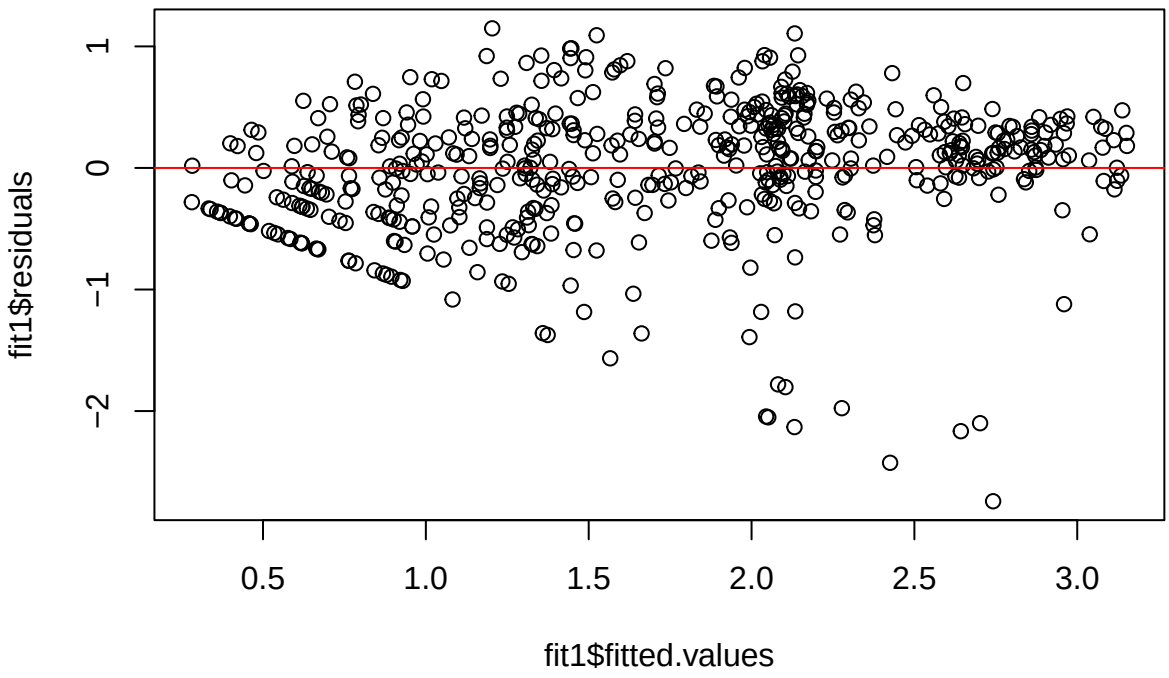


It is unfortunate but there isn't really any other values in our data set that we can really include in our spatial process fit. If I had time to do some web scrapes or look into some more data sets it would be worth looking to see if we can find data such as: charging stations within 20 miles, Urban or Rural, and Ev reduced Tax rate at location (some places incentivizes EV's with reduced Taxes).

```
fit1 <- spatialProcess(s, log10(y))
```

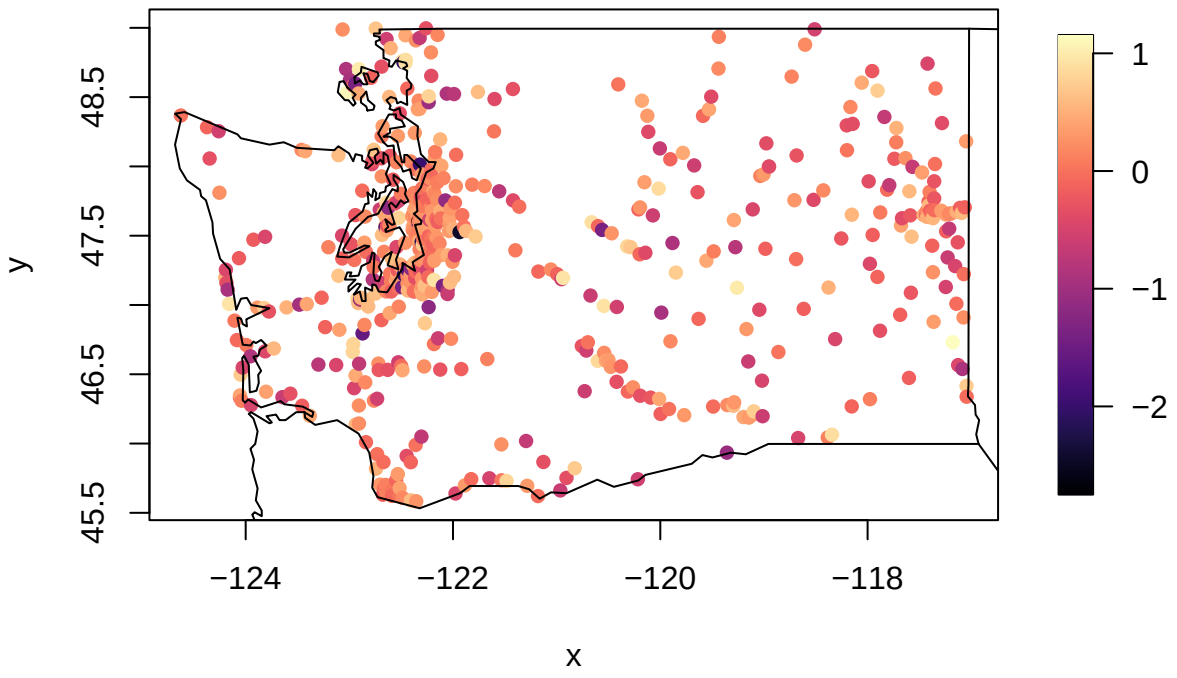
These logged EV counts residuals look way better than the non logged version. There is more spreading the farther away and it isn't perfect but I don't think any more transformations will help or beat this.

```
plot( fit1$fitted.values, fit1$residuals)
abline( h=0, col="red")
```



bubble plot of residuals. Can see where residuals are large and we aren't fitting well. It is important to remember that these values are in a log10 scale. Looks pretty well distributed. Errors are pretty speratically located and they seem to be near opposites of each other.

```
bubblePlot(s,fit1$residuals,  size=1, col=magma(256), highlight=FALSE)
#world( add=TRUE, col="grey", lwd=2)
US(add=TRUE)
```



Spatial Process has an default smoothness param of 1 so we are going to test with a exponential smoothness and a stronger smoothness and compare MLE estimates

```
fit1exp <- spatialProcess(s, log10(y),  smoothness =.5)
fit1Smooth <- spatialProcess(s, log10(y),  smoothness =1.5)
```

Looking at these values it seems as though the higher smoothness functions do better. This makes sense because we got some sparse locations which a good smoothness function will work better on. It seems as though the higher smoothness function is slightly better however I am going to fit an even higher smoothness function now to investigate further and then fit the residual vs fitted for all of these.

```
fit1$summary["lnProfileLike.FULL"]
```

```
## lnProfileLike.FULL
##          -573.6487
```

```
fit1$summary["lnProfileREML.FULL"]
```

```
## lnProfileREML.FULL
##          -579.2239
```

```
fit1exp$summary["lnProfileLike.FULL"]
```

```
## lnProfileLike.FULL
##          -575.4982
```

```
fit1exp$summary["lnProfileREML.FULL"]
```

```
## lnProfileREML.FULL
##          -580.8355
```

```
fit1Smooth$summary["lnProfileLike.FULL"]
```

```
## lnProfileLike.FULL
##          -573.4956
```

```
fit1Smooth$summary["lnProfileREML.FULL"]
```

```
## lnProfileREML.FULL
##          -579.1969
```

The larger smoothfunction doens't perform better than the 1.5 one. Technically the 1.5 does best according to the likelihood

```
fit1largeSmooth <- fit1Smooth <- spatialProcess(s, log10(y),  smoothness =2)
fit1largeSmooth$summary["lnProfileLike.FULL"]
```

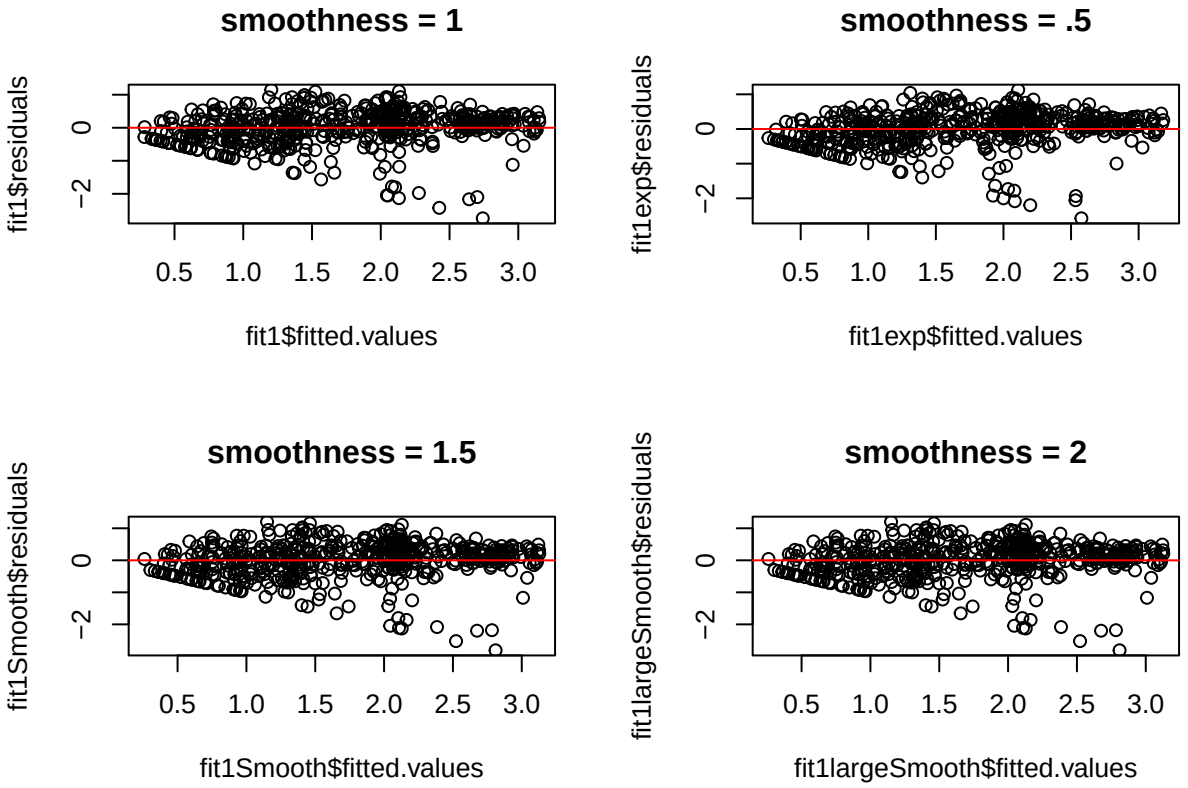
```
## lnProfileLike.FULL
##          -573.5707
```

```
fit1largeSmooth$summary["lnProfileREML.FULL"]
```

```
## lnProfileREML.FULL
##          -579.3514
```

Honestly all look really similar and I'm just gonna trust the likelihood and go with 1.5 but I don't think it'll make a huge difference.

```
par(mfrow = c(2, 2))
plot( fit1$fitted.values, fit1$residuals, main = "smoothness = 1")
abline( h=0, col="red")
plot( fit1exp$fitted.values, fit1exp$residuals, main = "smoothness = .5")
abline( h=0, col="red")
plot( fit1Smooth$fitted.values, fit1Smooth$residuals, main = "smoothness = 1.5")
abline( h=0, col="red")
plot( fit1largeSmooth$fitted.values, fit1largeSmooth$residuals, main = "smoothness = 2")
abline( h=0, col="red")
```



Creating Predictions

```
fitFinal <- fit1Smooth
```

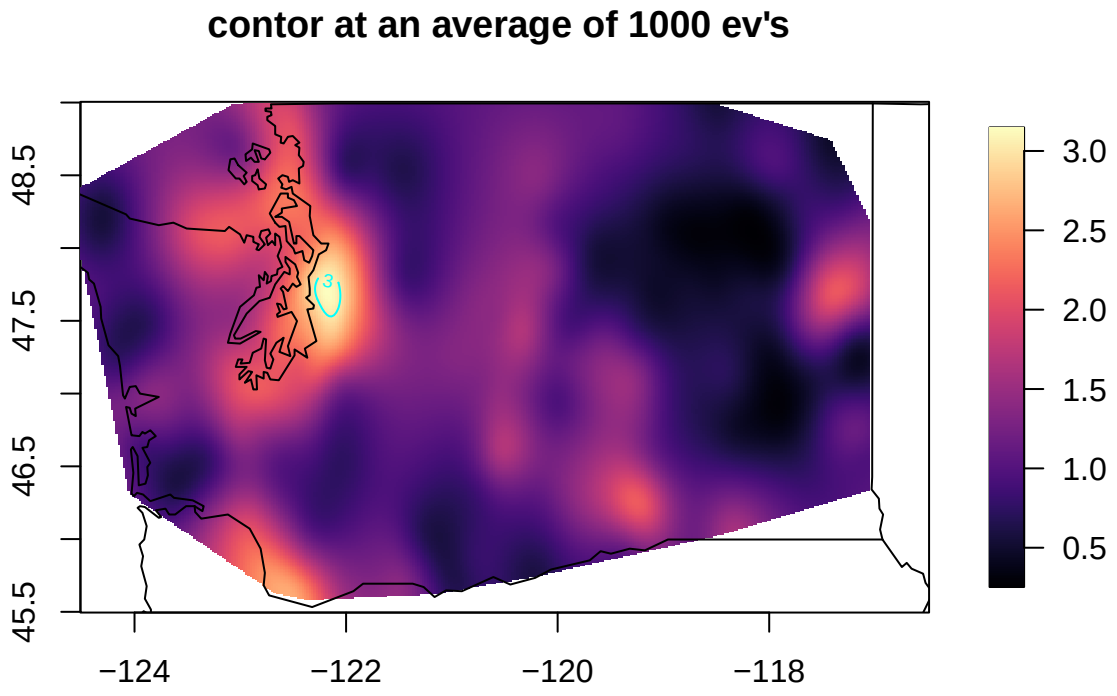
plotting surface

Did magma and turbo. I like magma because I think it looks cool but turbo is prob more useful. Remember the EV counts are logged

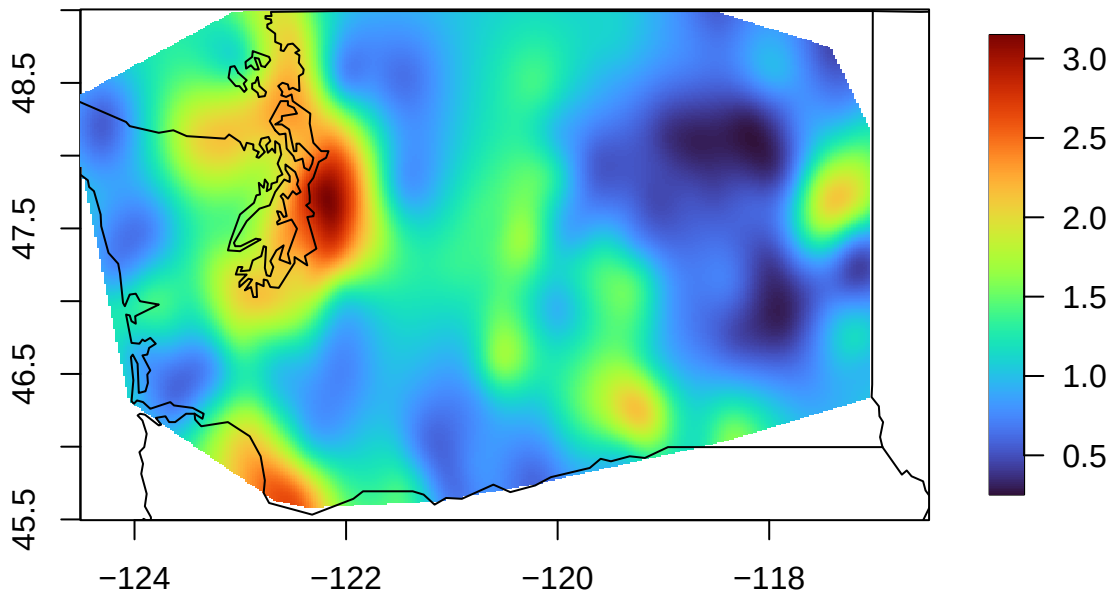
```
# number of points to predict over
N <- 360
M <- 360
#creating grid list
gridList <- list(
  x = seq(-124.5, -116.5, length.out = N),
  y = seq(45.5,49, length.out = M)
)

#predicting for points system to see how long it will take
system.time(
look<- predictSurface(fitFinal,gridList= gridList)
)
```

```
##    user  system elapsed
##   10.78    0.12   10.90
```

```
#its cool how it automatically gets rid of areas or points that we can't extrapolate to.
imagePlot(look, col = magma(256), main = "contor at an average of 1000 ev's ")
US( add=TRUE, col="black")
contour(look, add = T, levels = log10(1000), col = "cyan")
```




```
imagePlot(look, col = turbo(256))
US( add=TRUE, col="black")
```



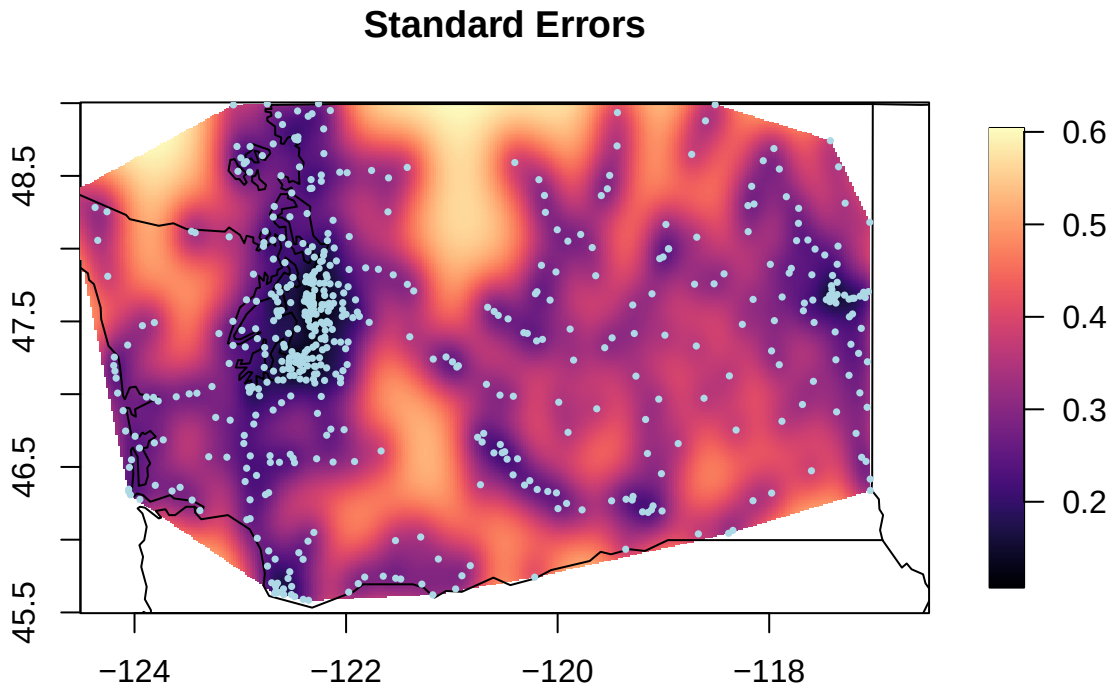
Finding out SE

```
system.time(
SE<- predictSurfaceSE( fitFinal, gridList)
)
```

```
##    user  system elapsed
##   54.00    0.42   54.42
```

Standard Errors of the predictions. Light blue points is where we have our data.

```
imagePlot(SE, col = magma(256), main = "Standard Errors")
US( add=TRUE, col="black")
bubblePlot(WashingtonEVs[,1:2],WashingtonEVs$evCount,  size=.5, col= "lightblue", highlight=FALSE, add =
```



Now we are going to focus more on Seattle which is the where there is definitely the most data.

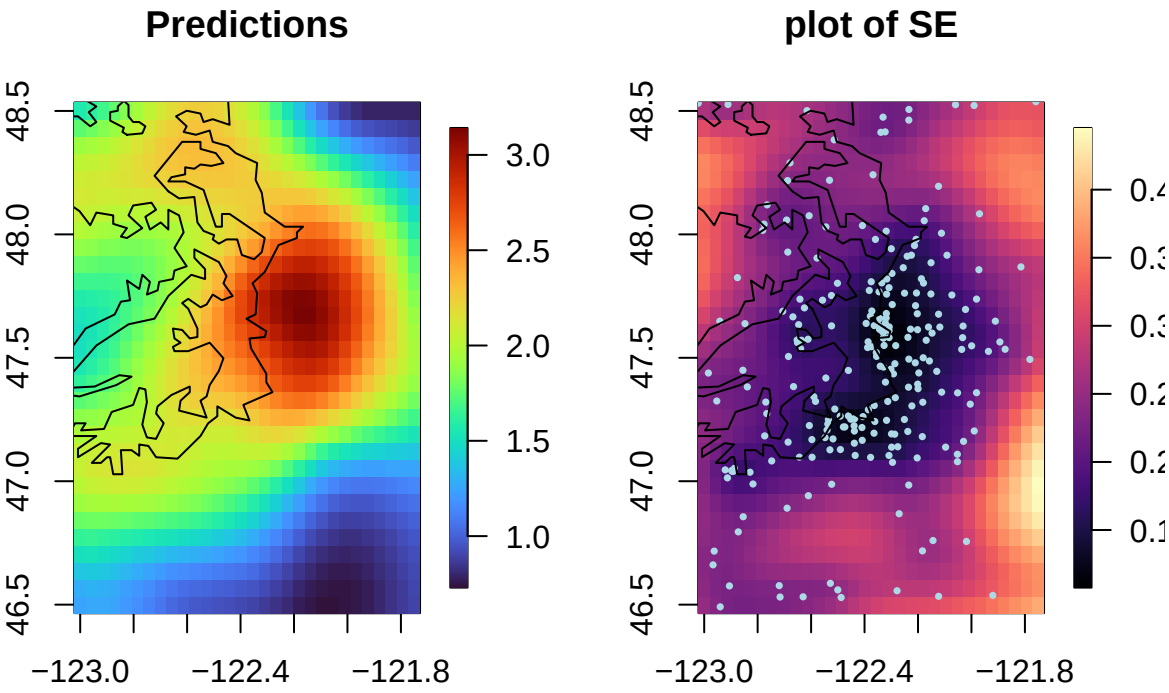
```
nGrid<- 30
gridList<- list( x = seq(-123, -121.75, length.out=nGrid),
                 y = seq(46.5, 48.5,length.out=nGrid ))
```

Find the prediction surface and prediction standard errors using

```
fHat<- predictSurface( fitFinal, gridList)
SE<- predictSurfaceSE( fitFinal, gridList)
```

plotting seattle results side by side

```
par(mfrow = c(1, 2))
imagePlot(fHat, col = turbo(256), main = "Predictions")
US( add=TRUE, col="black")

#I don't like having them in the same colorscale so I made the SE in magma
imagePlot(SE, col = magma(256), main = "plot of SE")
bubblePlot(WashingtonEVs[,1:2],WashingtonEVs$evCount,  size=.5, col= "lightblue", highlight=FALSE, add =
US( add=TRUE, col="black")
```


Predictions
48.5
48.0
47.5
47.0
46.5
−123.0
−122.4
−121.8
3.0
2.5
2.0
1.5
1.0
plot of SE
48.5
48.0
47.5
47.0
46.5
−123.0
−122.4
−121.8
0.4
0.3
0.3
0.2
0.2
0.1