Math 531 Exam 2

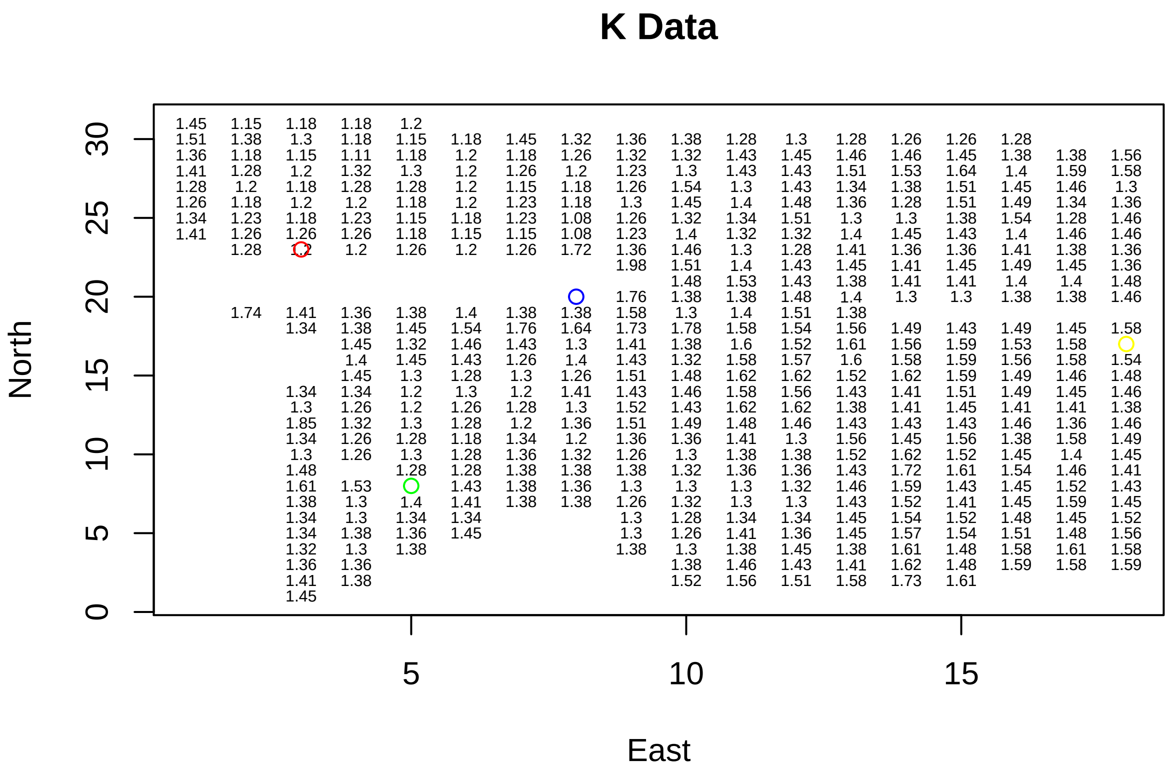
Gustavo Esparza

6/19/2019

Plot Of Original Data

Our objective for this assignment is to estimate which of the four plots ((3,23), (5,8), (8,20), and (18,17)) will have the most Potassium.

First, we will review a plot of the log potassium distribution over our grid.



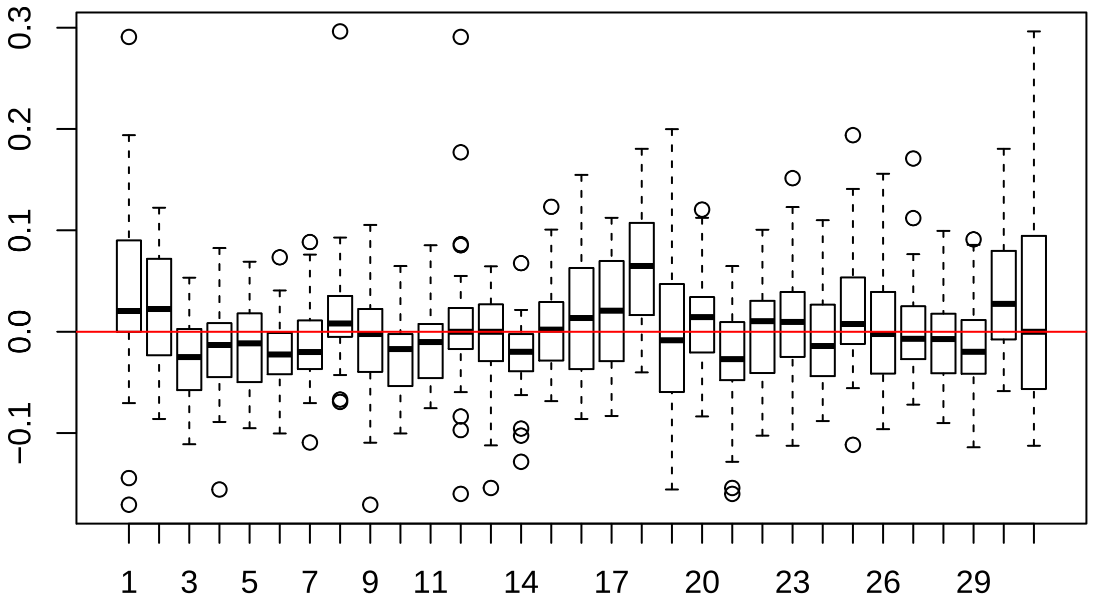
We wish to utilize this data to make predictions for four specific locations and determine which locations will yield the most potassium. From our observations, it seems that point 3 (8,20) and point 4 (18,17) reside in a location that contains the most potassium.

When beginning this objective, we have the initial choice of Median Polishing and Kernel Regression for our predictions. When deciding which route to take, it was important to have an objective value to compare between the two methods. Here, we utilized the sum of squared residuals for both methods. Included in ths file is a median polishing methodology (included in the Appendix) that yielded a larger SSR than the Kernel Regression that will be examined in this assignment.

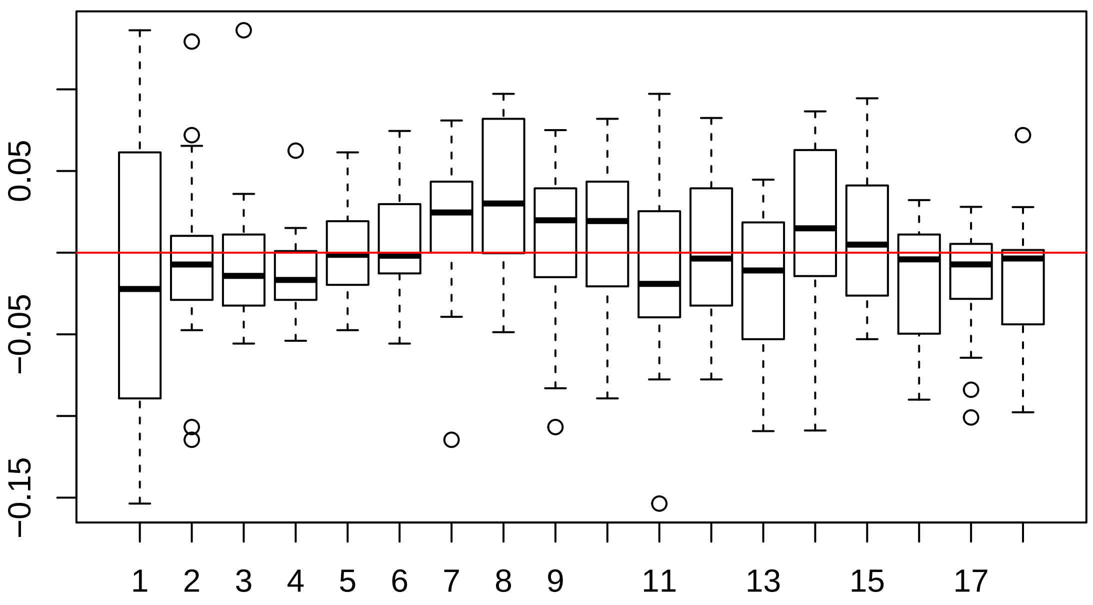
Pocket Plots

When attempting to forecast our log Potassium values, it's important to take into account any non-stationarity that may occur in the plot of our data. For this purpose, we will create Pocket Plots that provide a simple display of non-stationarity in our rows and columns.

Here are the two pocket plots for our East and North directions:



For our Eastern pocket plots, we can see that the majority of our East coordinates are stationary with a few exceptions that are questionable but not extreme deviations. East coordinate 18 is the largest violator of stationarity, but we will proceed and review the Northern coordinates.



Our Northern coordinates appear to be generally stationary, with no clear violators of this quality. Now that we have reviewed the stationarity of our data, we are able to continue with our forecasting methods. We will begin with a quick summary of the Median Polishing method.

Median Polishing

Via our Median Polishing function, we are able to take the overall, row and column effects for our data in order to create a robust model to fit values designated to a specific location.

Once we have created our Median Polishing Model, we can compute residuals and aqquire our SSR value that is used to decide which method (Median Polishing and Kernel Regression) is the most beneficial. here is that value:

```
## [1] 4.139621
```

We will see that this SSR value is larger than our Kernel Regression SSR, thus Median Polishing is not the optimal method for this dataset.

Before going into our Kernel Regression analysis, we will show that Median Polishing is able to make predictions for the four locations provided in the assignment. We can keep these results as a reference for the Kernel Regression results.

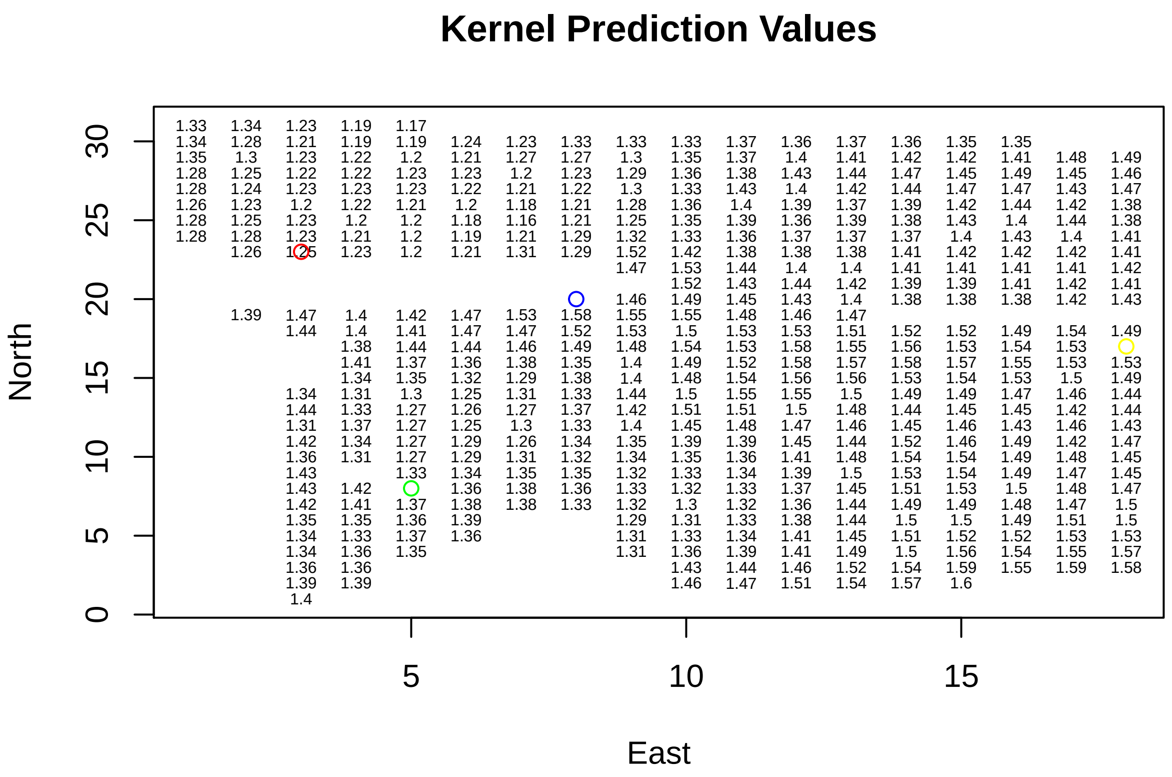
```
##  Predicted Potassium value for (3,23)  =  1.240403
##  Predicted Potassium value for (5,8)   =  1.335519
##  Predicted Potassium value for (8,20)  =  1.29073
##  Predicted Potassium value for (18,17) =  1.552996
```

We can see that our Median Polishing predicts (18,17) to be the best location for Potassium by a wide margin. Now that we have a reference point, we can utilize Kernel Regression to compare predictions.

Kernel Regression

We will begin be executing a Kernel Regression procedure and plotting the predicted values of each observed instance from our original data:

Here is a plot of our Kernel Predicted values:



This prediction plot does give similar values to our original data, but it also provides varying predictions for areas that pertain to our four choices. More closely, here are the four predicted values from our potential grid locations:

```
##  Predicted Potassium value for (3,23)  =  1.23027
##  Predicted Potassium value for (5,8)   =  1.390671
##  Predicted Potassium value for (8,20)  =  1.547118
##  Predicted Potassium value for (18,17) =  1.553349
```

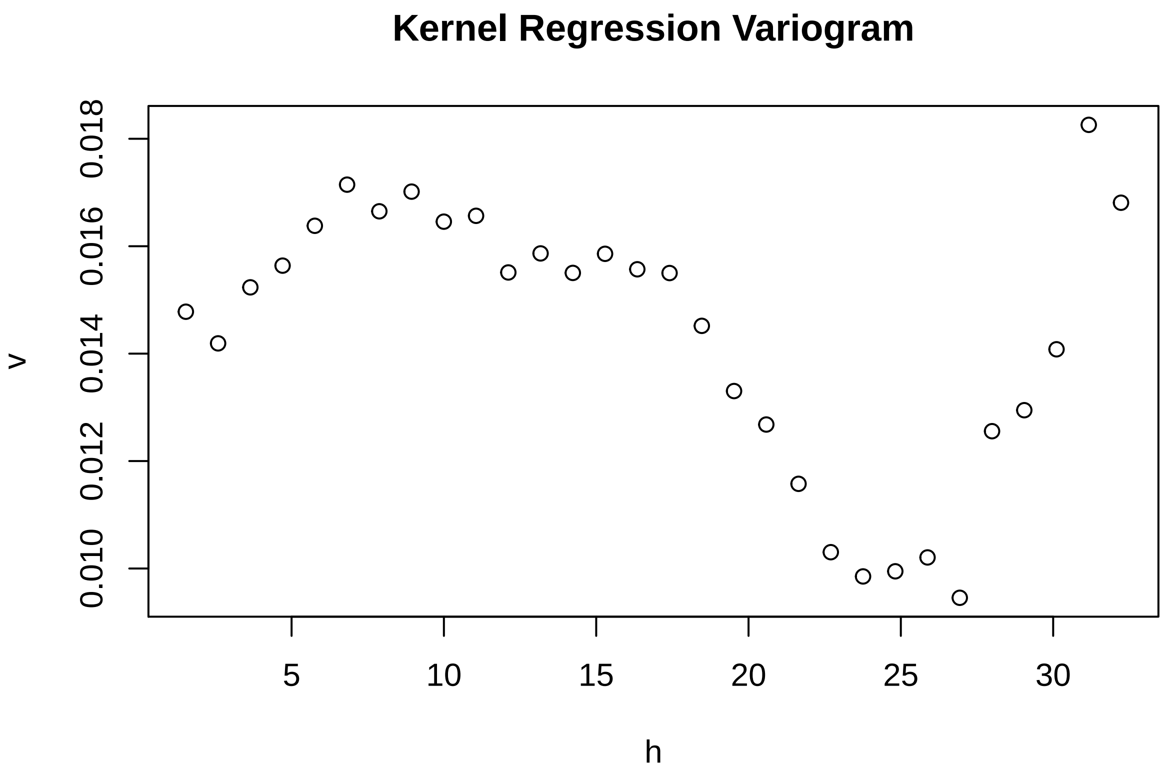
We can see that the last two locations are quite close to each other, so we would like to further investigate our model's residuals to see if there are any processes that can make the deciding choice more apparent. We will proceed to analyze our *residuals*, via variograms and detrministic models, to see if Kriging is able to determine the most reliable coordinate.

Before proceeding to our variogram and kriging procedure, we should observe our SSR value given via Kernel regression.
Here is the SSR when utilizing Kernel regression:

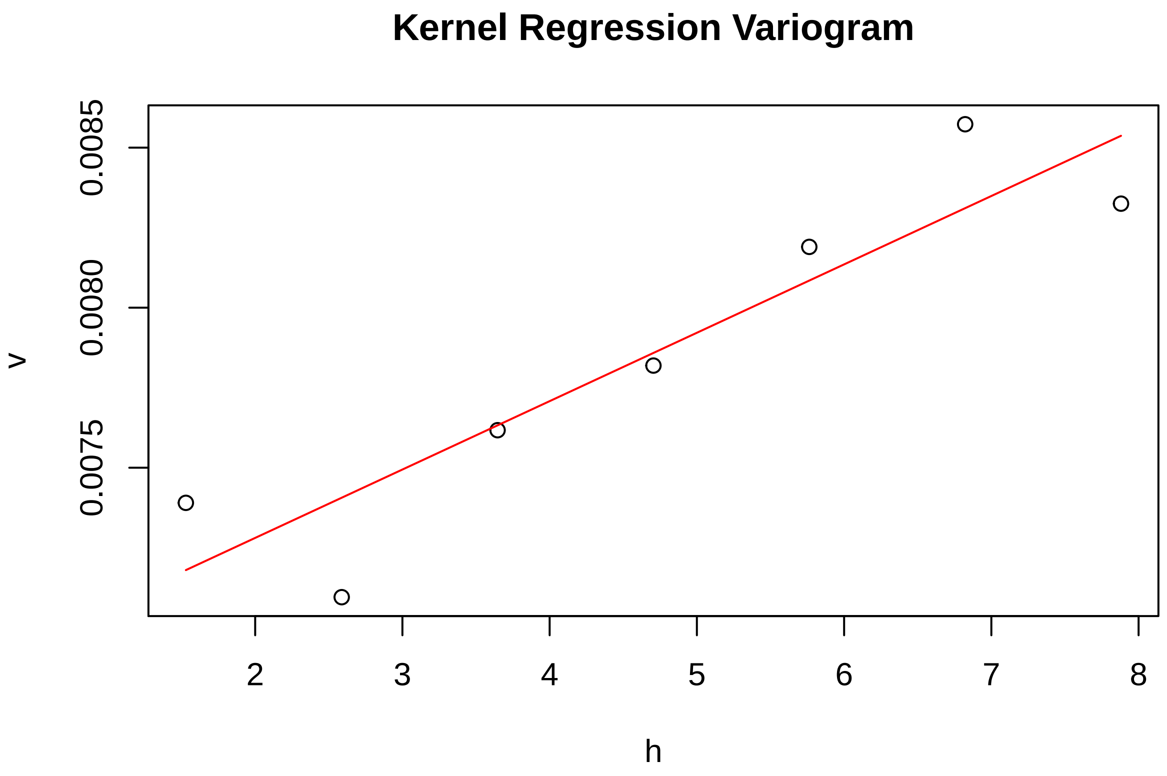
```
## SSR for Kernel regression =  3.297569
```

This SSR value is lower than our SSR value for the Median Polishing process (about 4), thus our decision for Kernel Regression is validated and we can proceed with our variograms.

Now, utilizing our Kernel Regression residuals, we will construct an omnidirectional variogram:



We can see that a optimal h value of 8 is appropriate for this variogram, before our graph loses a general trend and begins acting erraticly. Thus, we will restrict our variogram and also attempt to fit a linear model to our variogram.



Our linear model appears to fit the variogram data quite well, so we should be able to utilize the linear coeffecients for our kriging process.

Now that we have modeled out the deterministic aspect and fit an appropriate model to our residuals variogram, we can apply *kriging* to our Regression Model. Using the linear model obtained previously, we are able to use the coeffecients of the linear equation to compute predicted residual values that will benefit our kernel regression values.

Via our kriging process, we are able to obtain predicted values for the errors in the residual values for our previous Kernel regression predicted values. Thus, by adding the kiriging errors to our kernel regression predicted values, we will obtain updated predictions that have taken into account the error terms. Ultimately, these final prediction values are more reliable in reflecting the missing data points found in our plots.

Here are the updated predictions for Potassium aftr taking into effect our Kriging errors:

```
##  Kriging error value for (3,23)  =  -0.04588
##  Kriging error value for (5,8)   =  0.03253687
##  Kriging error value for (8,20)  =  0.08009201
##  Kriging error value for (18,17) =  0.05572346
##
```

```
##  Predicted Potassium value for (3,23)  =  1.18439
##  Predicted Potassium value for (5,8)   =  1.423208
##  Predicted Potassium value for (8,20)  =  1.62721
##  Predicted Potassium value for (18,17) =  1.609073
```

We can see that our kriging errors have been added to the Kernel Regression predictions, which have resulted in a wider margin in our third and fourth locations. We conclude that the third location, (8,20), contains the most predicted Potassium once Kriging errors are added to our Kernel predictions.

Appendix

Kernel Regression Code

We will begin be executing a Kernel Regression procedure and plotting the predicted values of each observed instance from our original data:

```r
Dij = matrix(0,n,n)
for(i in 1:n){
    for(j in 1:n){
        Dij[i,j] = sqrt((data$East[i] - data$East[j])^2 + (data$North[i] - data$North[j])^2)
    }
  }

shuffle.index = sample(1:n,n,replace=F)
test.index = vector("list",10)
index.n = floor(n/10)
    for(k in 1:10){
        test.index[[k]] = shuffle.index[(1+index.n*(k-1)):(index.n*k)]
    }

CVR2 = function(h){
prediction = c()
response = c()
for(k in 1:10){

    model.data = data[-test.index[[k]],]
    test.data = data[test.index[[k]],]
    model.n = length(model.data$East)
    test.n = length(test.data$East)

    Distance.mat = matrix(0,test.n,model.n)
    for(i in 1:test.n){
        for(j in 1:model.n){
            Distance.mat[i,j] = sqrt((model.data$East[j] - test.data$East[i])^2
                                         + (model.data$North[j] - test.data$North[i])^2)
        }
    }
    pred = rep(0,test.n)
    for(i in 1:test.n){
        pred[i] = sum(1/(sqrt(2*pi)*h^2)*exp(-Distance.mat[i,]^2/(2*h^2))*
            model.data$logK)/sum(1/(sqrt(2*pi)*h^2)*exp(-Distance.mat[i,]^2/(2*h^2)))
    }
    prediction = c(prediction,pred)
    response = c(response,test.data$logK)
}
SSR = sum((response - prediction)^2)
SSR
}

h=optim(2,CVR2)$par
h

Kern.Reg = rep(0,n)
```

```r
for(i in 1:n){
    Kern.Reg[i] = sum(1/(sqrt(2*pi)*h^2)*exp(-Dij[i,-i]^2/(2*h^2))*
        data$logK[-i])/sum(1/(sqrt(2*pi)*h^2)*exp(-Dij[i,-i]^2/(2*h^2)))
}
```

Here is a plot of our Kernel Predicted values:

```r
Kern.Reg=round(Kern.Reg,digits=2)
plot(data$East,data$North,type="n",xlab="East",ylab="North",main="Kernel Prediction Values")
text(data$East,data$North,Kern.Reg,cex=.5)
points(3,23,col="red")
points(5,8,col="green")
points(8,20,col="blue")
points(18,17,col="yellow")
```

SPECIFIC PREDICTIONS:

```r
data1 = data.frame(East = c(3,5,8,18), North = c(23,8,20,17))

Dij = matrix(0,4,n)
for(i in 1:4){
    for(j in 1:n){
        Dij[i,j] = sqrt((data1$East[i] - data$East[j])^2 + (data1$North[i] - data$North[j])^2)
    }}

Kern.Reg1 = rep(0,4)
for(i in 1:4){
    Kern.Reg1[i] = sum(1/(sqrt(2*pi)*h^2)*exp(-Dij[i,-i]^2/(2*h^2))*
      data$logK[-i])/sum(1/(sqrt(2*pi)*h^2)*exp(-Dij[i,-i]^2/(2*h^2)))
}
```

```r
cat(" Predicted Potassium value for (3,23)  = ",Kern.Reg1[1],
    "\n Predicted Potassium value for (5,8)   = ",Kern.Reg1[2],
    "\n Predicted Potassium value for (8,20)  = ",Kern.Reg1[3],
    "\n Predicted Potassium value for (18,17) = ",Kern.Reg1[4],"")
```

Here is the SSR when utilizing Kernel regression:

```r
data$resid = data$logK - Kern.Reg
resid=data$resid
ssr = sum(resid^2)
cat("SSR for Kernel regression = ",ssr,"")
```

VARIOGRAM:

```r
h.var = function(x,y,data,dir = 1){
    n = length(x)
    x.dist = c()
    y.dist = c()
    Z.diff.sq = c()
    xin = c()
    xout = c()
    yin = c()
    yout = c()
    for(i in 1:n){
        x.dist = c(x.dist,abs(x[i] - x[-i]))
        y.dist = c(y.dist,abs(y[i] - y[-i]))
        Z.diff.sq = c(Z.diff.sq,(data[i] - data[-i])^2)
        xin = c(xin,x[rep(i,n-1)])
        xout = c(xout,x[(1:n)[-i]])
        yin = c(yin,y[rep(i,n-1)])
        yout = c(yout,y[(1:n)[-i]])
    }
    if(dir==1){
        h = sqrt(x.dist^2 + y.dist^2)
    } else{
        if(dir == 2){
            h = x.dist[yin==yout]
            Z.diff.sq = Z.diff.sq[yin==yout]}
        else{
            h=y.dist[xin==xout]
            Z.diff.sq = Z.diff.sq[xin==xout]
        }
             }
    output = data.frame(h=h,Z.diff.sq=Z.diff.sq)
    output
}

k.variogram = function(x,y,data,dir=1,n.bins=30){
    h = h.var(x,y,data,dir=dir)$h
    diff = h.var(x,y,data,dir=dir)$Z.diff.sq
    bin.size = (max(h)-min(h))/n.bins
    gamma2 = rep(0,n.bins)
    n.per.bin = rep(0,n.bins)
    h.stag = seq(min(h)+.5*bin.size,max(h)-.5*bin.size,bin.size)
    bin.min = seq(min(h),max(h)-1*bin.size,bin.size)
    bin.max = seq(min(h) + bin.size,max(h),bin.size)
    for(i in 1:(n.bins-1)){
        n.per.bin[i] = length(h[h<bin.max[i]&h>=bin.min[i]])
        gamma2[i] = sum(diff[h<bin.max[i]&h>=bin.min[i]])/n.per.bin[i]
    }
    n.per.bin[n.bins] = length(h[h<=bin.max[n.bins]&h>=bin.min[n.bins]])
    gamma2[n.bins] = sum(diff[h<=bin.max[n.bins]&h>=bin.min[n.bins]])/n.per.bin[n.bins]

    output = data.frame(h = h.stag,gamma2 = gamma2)
    output}
```

```
out4 = k.variogram(data$East,data$North,resid)
plot(out4$h,out4$gamma2,xlab="h",ylab="v",main="Kernel Regression Variogram")
```

RESTRICTED VARIOGRAM

```
out5 = out4[out4$h<8,]

h=out5$h
gamma = out5$gamma2/2
g = lm(gamma~h)

plot(h,gamma,xlab="h",ylab="v",main="Kernel Regression Variogram")
a = min(out5$h)
b = max(out5$h)
```

```
lines(c(a,b),c(sum(g$coef*c(1,a)),sum(g$coef*c(1,b))),col=2)
```

KRIGING:

```
#distance of all pairs of points
hij = matrix(0,length(data$logK),length(data$logK))
for(i in 1:length(data$logK)){
  for(j in 1:length(data$logK)){
    hij[i,j] = sqrt((data$East[i] - data$East[j])^2 + (data$North[i] - data$North[j])^2)
  }
}

#distances from 4 points of interest to all other points
data1=data.frame(East = c(3,5,8,18),North = c(23,8,20,17))
h0j=matrix(0,4,length(data$logK))
for(i in 1:4){
  for(j in 1:length(data$logK)){
    h0j[i,j] = sqrt((data1$East[i] - data$East[j])^2 + (data1$North[i] - data$North[j])^2)
  }
}

gij = hij*b + a
g0j=h0j*b+a
one.vec = as.matrix(rep(1,length(data$logK)),ncol=1)
```

```
predictions = rep(0,4)
lambda.list = vector("list",4)

for(i in 1:4){
  g0i=matrix(t(g0j)[,i],ncol=1)
  lambda = t(g0i + one.vec%*%((1 - t(one.vec)%*%solve(gij)%*%g0i)/
                                (t(one.vec)%*%solve(gij)%*%one.vec)))%*%solve(gij)
  lambda.list[[i]]=lambda

  m = -(1 - t(one.vec)%*%solve(gij)%*%g0i)/(t(one.vec)%*%solve(gij)%*%one.vec)

  predictions[i] = sum(lambda*data$res)
}
```

Via our kriging process, we are able to obtain predicted values for the errors in the residual values for our previous Kernel regression predicted values. Thus, by adding the predicted values to our kernel regression predicted values, we will obtain updated predictions that have taken into account the error terms. Ultimately, these final prediction values are more reliable in reflecting the missing data points found in our plots.

Here are the updated predictions for Potassium aftr taking into effect our Kriging errors:

```
estimated_logk=Kern.Reg1+predictions


cat(" Predicted Potassium value for (3,23)  = ",estimated_logk[1],
    "\n Predicted Potassium value for (5,8)   = ",estimated_logk[2],
    "\n Predicted Potassium value for (8,20)  = ",estimated_logk[3],
    "\n Predicted Potassium value for (18,17) = ",estimated_logk[4],"")
```

Median Polishing Code

POCKET PLOTS:

```r
n1 = max(data$East)
n2 = max(data$North)
zz = matrix(NA,nrow=n1,ncol=n2)
for(i in 1:length(data$East)){
    zz[data$East[i],data$North[i]]= data$logK[i]
}
```

```r
pocket.plot.row=function(data){
    n1 = dim(data)[1]
    zz=data
    n.mat = matrix(0,nrow=n1,ncol=n1)
    y.mat = matrix(0,nrow=n1,ncol=n1)
    for(i in c(1:(n1-1))){
        for(j in c((i+1):n1)){
            n.mat[i,j] = length(na.omit(zz[i,]-zz[j,]))
            n.mat   [j,i] = length(na.omit(zz[i,]-zz[j,]))
            y.mat[i,j] = mean(sqrt(abs(na.omit(zz[i,] - zz[j,]))))
            y.mat[j,i] = y.mat[i,j]
            }
        }
    y.dbl.bar = rep(0,(n1-1))
    for(k in c(1:(n1-2))){
        temp=c()
        temp.n=c()
        for(i in c(1:(n1-k))){
            if(n.mat[i,(i+k)]>0){
            temp.n = c(temp.n,n.mat[i,(i+k)])
            temp = c(temp,y.mat[i,(i+k)])
            }
        }
    y.dbl.bar[k] = sum(temp*temp.n)/sum(temp.n)
    }
    y.dbl.bar[n1-1] = y.mat[1,n1]

    p = matrix(rep(0,(n1*n1)),ncol=n1)
    for(i in c(1:(n1-1))){
        for(j in c((i+1):n1)){
            k = j-i
            p[i,j] = y.mat[i,j] - y.dbl.bar[k]
        p[j,i] = p[i,j]
        }
    }
    p}

pocket.plot.col=function(data){
    n2 = dim(data)[2]
    zz=data
    n.mat = matrix(0,nrow=n2,ncol=n2)
    y.mat = matrix(0,nrow=n2,ncol=n2)
    for(i in c(1:(n2-1))){
        for(j in c((i+1):n2)){
            n.mat[i,j] = length(na.omit(zz[,i]-zz[,j]))
            n.mat   [j,i] = length(na.omit(zz[,i]-zz[,j]))
            y.mat[i,j] = mean(sqrt(abs(na.omit(zz[,i] - zz[,j]))))
            y.mat[j,i] = y.mat[i,j]
            }
        }

    y.dbl.bar = rep(0,(n2-1))
    for(k in c(1:(n2-2))){
        temp=c()
        temp.n=c()
        for(i in c(1:(n2-k))){
            if(n.mat[i,(i+k)]>0){
            temp.n = c(temp.n,n.mat[i,(i+k)])
            temp = c(temp,y.mat[i,(i+k)])
            }
        }
    y.dbl.bar[k] = sum(temp*temp.n)/sum(temp.n)
    }
    y.dbl.bar[n2-1] = y.mat[1,n2]

    p = matrix(rep(0,(n2*n2)),ncol=n2)
    for(i in c(1:(n2-1))){
        for(j in c((i+1):n2)){
            k = j-i
            p[i,j] = y.mat[i,j] - y.dbl.bar[k]
        p[j,i] = p[i,j]
        }
    }
    p}
```

```r
p=pocket.plot.col(zz)
boxplot(na.omit(p[,1]),na.omit(p[,2]),na.omit(p[,3]),na.omit(p[,4]),na.omit(p[,5]),
        na.omit(p[,6]),
        na.omit(p[,7]),na.omit(p[,8]),na.omit(p[,9]),na.omit(p[,10]),na.omit(p[,11]),
        na.omit(p[,12]),
        na.omit(p[,13]),na.omit(p[,14]),na.omit(p[,15]),na.omit(p[,16]),na.omit(p[,17]),
        na.omit(p[,18]),
        na.omit(p[,19]),na.omit(p[,20]),na.omit(p[,21]),na.omit(p[,22]),na.omit(p[,23]),
        na.omit(p[,24]),
        na.omit(p[,25]),na.omit(p[,26]),na.omit(p[,27]),na.omit(p[,28]),na.omit(p[,29]),
        na.omit(p[,30]),
        na.omit(p[,31]) )
abline(0,0,col=2)


p=pocket.plot.row(zz)
boxplot(na.omit(p[,1]),na.omit(p[,2]),na.omit(p[,3]),na.omit(p[,4]),
        na.omit(p[,5]),na.omit(p[,6]),
 na.omit(p[,7]),na.omit(p[,8]),na.omit(p[,9]),na.omit(p[,10]),na.omit(p[,11]),na.omit(p[,12]),
na.omit(p[,13]),na.omit(p[,14]),na.omit(p[,15]),na.omit(p[,16]),na.omit(p[,17]),na.omit(p[,18]))
abline(0,0,col=2)
```

MEDIAN POLISHING:

```r
n1 = max(data$East)
n2 = max(data$North)
zz = matrix(NA,nrow=n1,ncol=n2)
for(i in 1:length(data$East)){
    zz[data$East[i],data$North[i]]= data$logK[i]
}
```

```r
na.median = function(vector){
    median(na.omit(vector))
}

med.polish = function(data,k){
    n1 = length(data[,1])
    n2 = length(data[1,])
    col1 = apply(data,1,na.median)
    data1 = data - col1
    row1 = apply(data1,2,na.median)
    data2 = data1 - matrix(rep(row1,n1),ncol=n2,byrow=T)
    overall2 = na.median(col1)
    col2 = col1 - na.median(col1)
    col3 = col2
    row3 = row1
    for(i in 2:k){
        col1 = apply(data2,1,na.median)
        data1 = data2 - col1
        row1 = row3 - na.median(row3)
        overall1 = overall2 + na.median(row3)
        col2 = col3 + col1
        row2 = apply(data1,2,na.median)
        data2 = data1 - matrix(rep(row2,n1),ncol=n2,byrow=T)
        row3 = row1 + row2
        overall2 = overall1 + na.median(col2)
        col3 = col2 - na.median(col2)
    }
    return(list(data2,col3,row3,overall2))
}
```

Computing Median Polishing Residuals:

```r
polish = med.polish(zz,100)
residuals = polish[[1]]
```

```r
data$residuals=0

for(i in 1:18){
  for(j in 1:31){
    if(!(is.na(residuals[i,j]) ) ){
      data$residuals[data$East==i&data$North==j] = residuals[i,j]
    }}}
```

```r
#Median Polishing SSR = 4.14
resid = data$residuals
r2 = sum(resid^2)
r2
```

We are able to see that the SSR value is reduced when utlizing Kernel Regression as opposed to Median

Polishing.

PREDICTING VALUES:

```
#Median Polishing Predictions


predictions=polish[[4]]+polish[[3]][data$North]+polish[[2]][data$East]
Z_s=matrix(NA,nrow=n1,ncol=n2)
for(i in 1:length(data$East)){Z_s[data$East[i],data$North[i]]=predictions[i]}
```

```
fit=function(east,north){
  z_ij=polish[[4]]+polish[[3]][north]+polish[[2]][east]
  return(z_ij)
}
```

At this point, SSR values were compared for the two methods and Kernel Regression was decided to be the best option for Forecasting and Deterministic Modeling.