Comparison of k-NN, Weighted k-NN, and Weighted Local Linear Regression

Alexandre H.

```r
library(tidyverse)

set.seed(20231101)

n <- 300
x <- rnorm(n, 3, 1)
e <- rnorm(n, 0, 1)
y <- 0.5 + 0.1 * x + 0.2 * x^2 + e

m <- 300
x_test <- rnorm(m, 3, 1)
e_test <- rnorm(m, 0, 1)
y_test <- 0.5 + 0.1 * x_test + 0.2 * x_test^2 + e_test

x_grid <- seq(0, 6, by=0.02)
```

Part 1: k-NN Regression

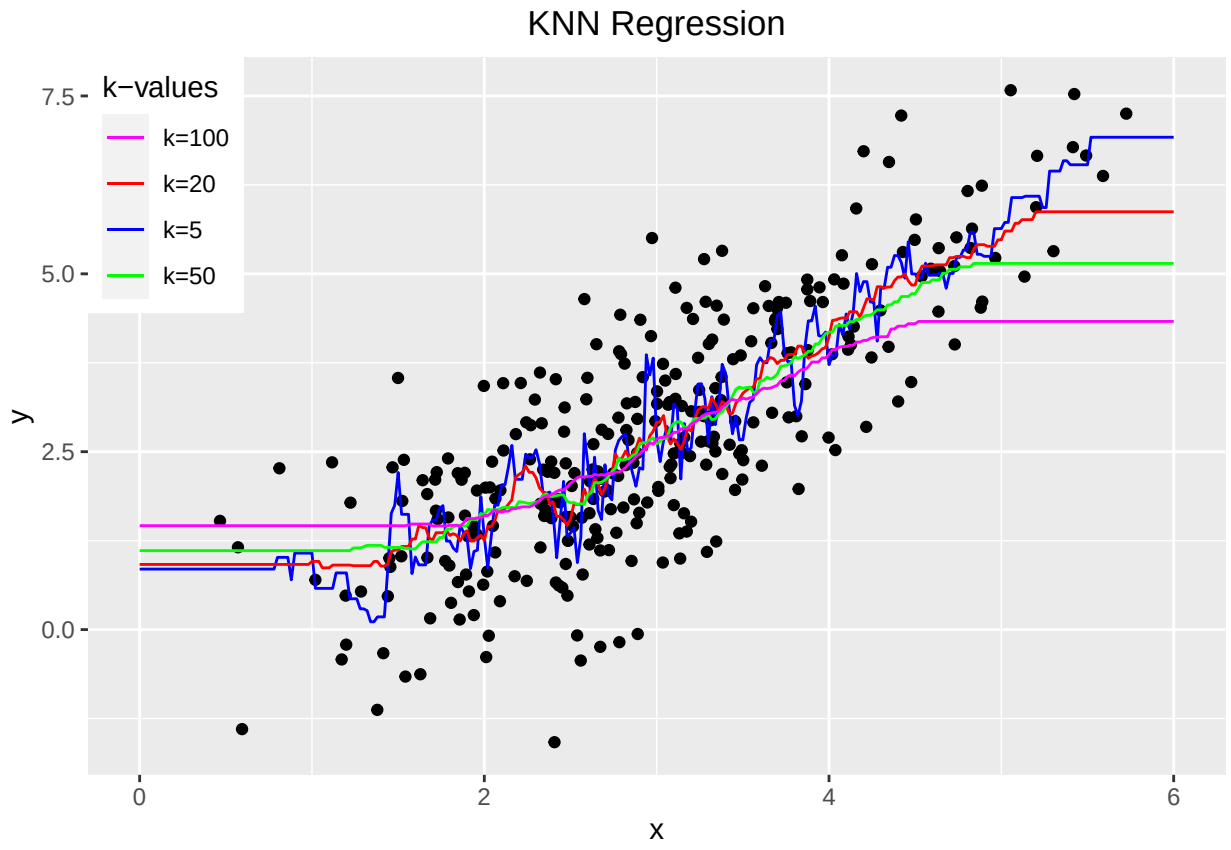
```r
simple_knn <- function(k, x_0, x_data, y_data){
"Returns the k-NN regression prediction for a single x_0 value"
distances <- abs(x_data - x_0)
nearest_indices <- order(distances)[1:k]
return(mean(y_data[nearest_indices]))
}

k_values <- c(5, 20, 50, 100)
x_grid_predictions <- matrix(NA, nrow=length(x_grid), ncol=length(k_values))
for (i in seq_along(k_values)){
  x_grid_predictions[,i] <- map_dbl(x_grid, ~simple_knn(k_values[i], .x, x, y))
}

colors <- c("k=5" = "#0000FF", "k=20" = "#FF0000",
            "k=50" = "#00FF00", "k=100" = "#FF00FF")

ggplot(data.frame(x=x, y=y), aes(x=x, y=y)) +
geom_point() +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,1], color=paste0("k=",k_values[1]))) +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
```

```
aes(x=x, y=x_grid_predictions[,2], color=paste0("k=",k_values[2]))) +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,3], color=paste0("k=",k_values[3]))) +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,4], color=paste0("k=",k_values[4]))) +
labs(x="x", y="y", title="KNN Regression", color="k-values") +
theme(plot.title=element_text(hjust=0.5)) +
scale_color_manual(values = colors) +
theme(legend.position = c(0, 1), legend.justification = c(0, 1))
```



Looking at the k-NN regression results, the model's behavior significantly varies with different k values. When $k = 5$, the prediction line is very wavy, adhering closely to the individual data points. This pattern suggests the model might be too focused on the training data's specific details, including noise, which is not ideal.

As k increases to 20, the prediction line becomes less wavy. It's not following every up and down of the training data, suggesting it's getting better at ignoring the noise and focusing on the overall trend.

At $k = 50$, the line smoothens out more, indicating that the model is becoming even better at generalizing. It starts to overlook the random noise in the data, which is a positive sign of a more reliable model.

When k is up at 100, the prediction line is the smoothest. This level of smoothness could be a sign that the model is now too general and might be missing important patterns in the data, which could lead to underfitting.

In summary, a small k can cause the model to be overly sensitive to noise, while a large k can lead to overlooking important data patterns. It seems like a value of k between 20 and 50 might offer a good balance to capture the trend without fitting the noise.

Part 2: Weighted k-NN Regression

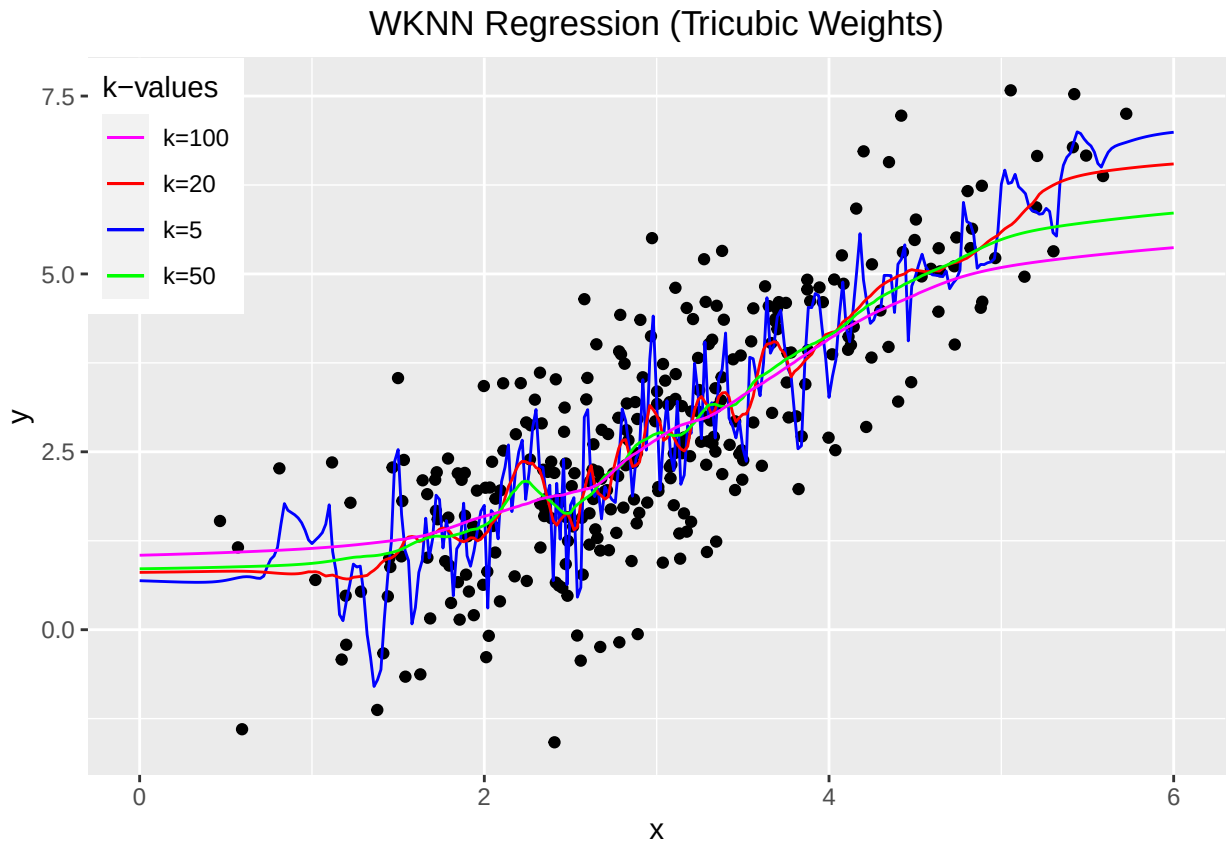
```
wknn_tricubic <- function(k, x_0, x_data, y_data){
"Returns the weighted k-NN regression prediction for a
single x_0 value using tricubic weights"
distances <- abs(x_data - x_0)
nearest_indices <- order(distances)[1:k]
k_neighboors <- x_data[nearest_indices]
w_til <- (1-(abs(k_neighboors - x_0) / max(abs(k_neighboors - x_0)))^3)^3
w <- w_til / sum(w_til)
return(sum(w * y_data[nearest_indices]))
}

k_values <- c(5, 20, 50, 100)
x_grid_predictions <- matrix(NA, nrow=length(x_grid), ncol=length(k_values))
for (i in seq_along(k_values)){
  x_grid_predictions[,i] <- map_dbl(x_grid, ~wknn_tricubic(k_values[i], .x, x, y))
}



colors <- c("k=5" = "#0000FF", "k=20" = "#FF0000",
            "k=50" = "#00FF00", "k=100" = "#FF00FF")

ggplot(data.frame(x=x, y=y), aes(x=x, y=y)) +
geom_point() +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,1], color=paste0("k=",k_values[1]))) +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,2], color=paste0("k=",k_values[2]))) +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,3], color=paste0("k=",k_values[3]))) +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,4], color=paste0("k=",k_values[4]))) +
labs(x="x", y="y", title="WKNN Regression (Tricubic Weights)", color="k-values") +
theme(plot.title=element_text(hjust=0.5)) +
scale_color_manual(values = colors) +
theme(legend.position = c(0, 1), legend.justification = c(0, 1))
```



The weighted k-nearest neighbor regression with tricubic weights is applied to the simulated data for different values of k. The behavior of the model with respect to these values is as follows:

- For $k = 5$: The prediction line shows sensitivity to the training data, but less than the unweighted version. The tricubic weighting reduces the influence of noise by giving more importance to the nearest neighbors. It leads to a somewhat wavy prediction line.
- For $k = 20$: The prediction curve suggests a good balance in capturing the underlying data structure. It smoothens out noise effectively while still adapting to local data variations. The weighted scheme emphasizes closer points more significantly.
- For $k = 50$: Further smoothing is observed in the prediction line, which indicates a strong generalization of the data. The model integrates a wider neighborhood while still prioritizing closer points. This results in a stable and generalized prediction that reflects the main trends without overfitting to noise.
- For $k = 100$: The smoothest prediction line among the different values of k is observed here. Despite the benefits of the weighting, the large k value may lead to a weaker impact from the nearest neighbors. This can cause the model to underfit and potentially miss certain patterns in the data.

So we can observe that the weighted k-NN with tricubic weights generally provides smoother predictions compared to the unweighted k-NN. By observation, it appears that a k value between 20 and 50 might be a good balance.

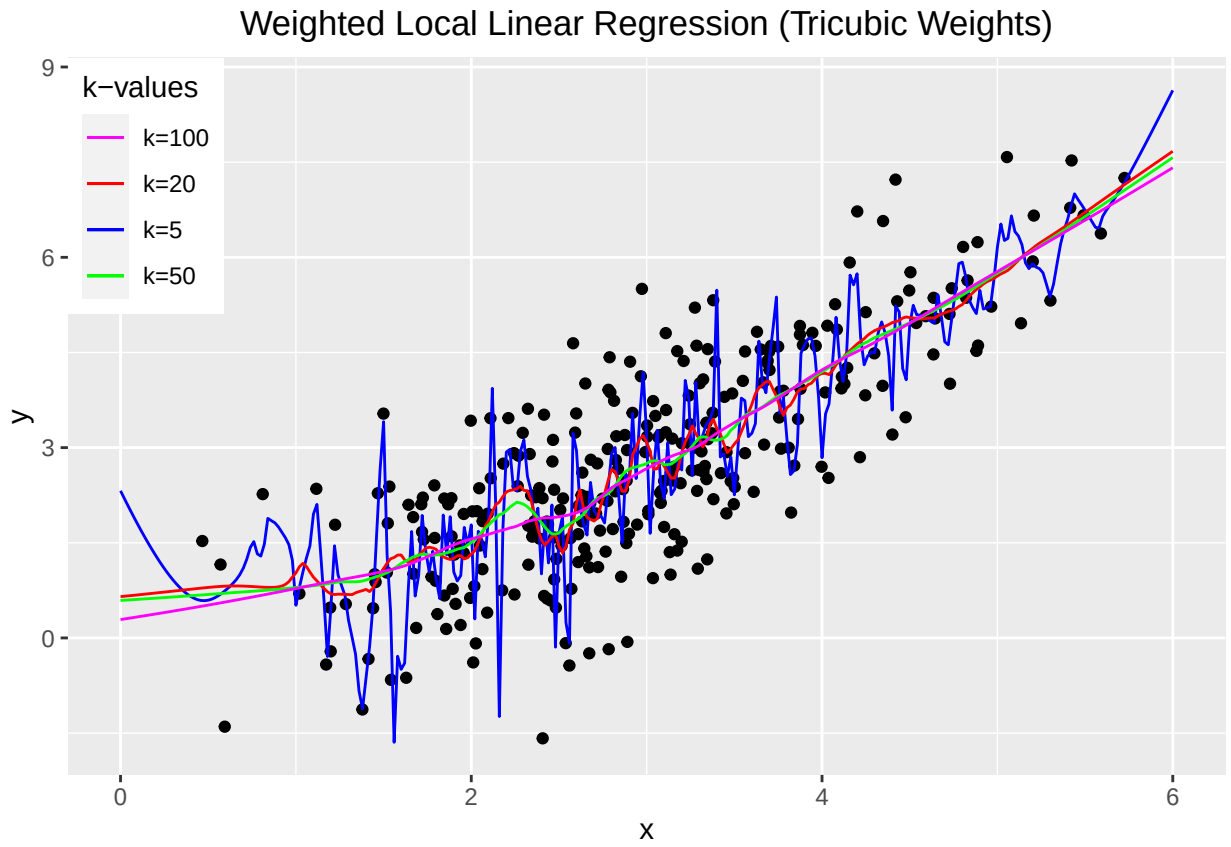
Part 3: Weighted Local Linear Regression

```
weighted_local_lm <- function(k, x_0, x_data, y_data){
"Returns the prediction of a local linear model with tricubic weights"
distances <- abs(x_data - x_0)
nearest_indices <- order(distances)[1:k]
k_neighboors <- x_data[nearest_indices]
w_til <- (1-(abs(k_neighboors - x_0) / max(abs(k_neighboors - x_0)))^3)^3
w <- w_til / sum(w_til)
w_lm <- lm(y_data[nearest_indices] ~ x_data[nearest_indices], weights=w)
beta_0 <- w_lm$coefficients[1]
beta_1 <- w_lm$coefficients[2]
return(beta_0 + beta_1 * x_0)
}


k_values <- c(5, 20, 50, 100)
x_grid_predictions <- matrix(NA, nrow=length(x_grid), ncol=length(k_values))
for (i in seq_along(k_values)){
  x_grid_predictions[,i] <- map_dbl(x_grid, ~weighted_local_lm(k_values[i], .x, x, y))
}


colors <- c("k=5" = "#0000FF", "k=20" = "#FF0000",
            "k=50" = "#00FF00", "k=100" = "#FF00FF")

ggplot(data.frame(x=x, y=y), aes(x=x, y=y)) +
geom_point() +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,1], color=paste0("k=",k_values[1]))) +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,2], color=paste0("k=",k_values[2]))) +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,3], color=paste0("k=",k_values[3]))) +
geom_line(data=data.frame(x=x_grid, y=x_grid_predictions),
aes(x=x, y=x_grid_predictions[,4], color=paste0("k=",k_values[4]))) +
labs(x="x", y="y",
     title="Weighted Local Linear Regression (Tricubic Weights)", color="k-values") +
theme(plot.title=element_text(hjust=0.5)) +
scale_color_manual(values = colors) +
theme(legend.position = c(0, 1), legend.justification = c(0, 1))
```



The performance of the weighted local linear regression with tricubic weights is evaluated for different values of k. The behavior of the regression model for these values is as follows:

- For $k = 5$: The model's predictions are quite volatile, particularly at the extremities. The local linear models are very responsive to the small sample of neighbors, leading to a high variance. It is most likely indicating overfitting.
- For $k = 20$: The prediction curve shows less fluctuation compared to $k = 5$, suggesting an improvement. With more neighbors, the impact of noise and outliers is diminished. This results in a more stable estimation that keeps some local data sensitivity.
- For $k = 50$: The resulting curve is smoother and demonstrates stability across the range. The local linear models now incorporate a larger neighbor sample. It smoothens out the variance even more than the lower previous values of k. Although the variance of the model has now decreased, there is a risk of having increased the bias too much, which can lead to underfitting.
- For $k = 100$: The curve is the smoothest, showing a significant reduction in the influence of individual data points. This level of smoothness implies that the model primarily captures the main trend, but may again be at risk of underfiting just like for a value of $k = 50$ by not accounting for smaller variations.

Thus, as k increases, the model transitions from fitting closely to local data nuances risking overfiting to emphasizing the overall data trend by risking underfiting. A good k value requires to balance variance against bias. Visually, it seems like a value of k between 20 and 50 might be a good balance.

Part 4: MSE on Test Data

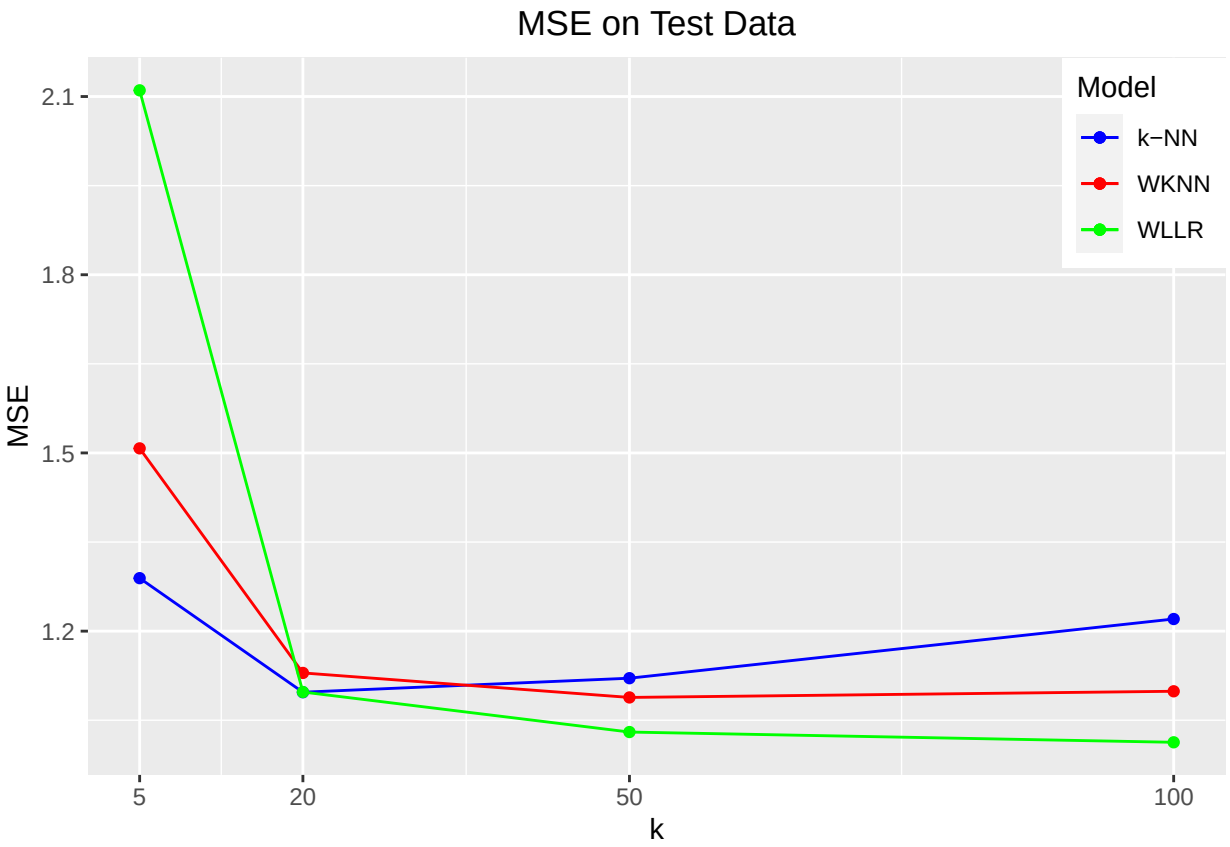
```
k_values <- c(5, 20, 50, 100)
mse_test_all <- matrix(NA, nrow=length(k_values), ncol=3)

for (i in seq_along(k_values)){
y_pred <- map_dbl(x_test, ~simple_knn(k_values[i], .x, x, y))
mse_test_all[i,1] <- mean((y_test - y_pred)^2)
y_pred <- map_dbl(x_test, ~wknn_tricubic(k_values[i], .x, x, y))
mse_test_all[i,2] <- mean((y_test - y_pred)^2)
y_pred <- map_dbl(x_test, ~weighted_local_lm(k_values[i], .x, x, y))
mse_test_all[i,3] <- mean((y_test - y_pred)^2)
}

colnames(mse_test_all) <- c("k-NN", "WKNN", "WLLR")
rownames(mse_test_all) <- k_values
mse_test_all
```

```
##         k-NN     WKNN     WLLR
## 5   1.289169 1.507692 2.110448
## 20  1.097174 1.129623 1.097513
## 50  1.120770 1.088379 1.030132
## 100 1.220415 1.098750 1.012852
```

```
ggplot(data.frame(k=k_values, mse_test_all), aes(x=k, y=mse_test_all)) +
geom_line(aes(x=k, y=mse_test_all[,1], color="k-NN")) +
geom_point(aes(x=k, y=mse_test_all[,1], color="k-NN")) +
geom_line(aes(x=k, y=mse_test_all[,2], color="WKNN")) +
geom_point(aes(x=k, y=mse_test_all[,2], color="WKNN")) +
geom_line(aes(x=k, y=mse_test_all[,3], color="WLLR")) +
geom_point(aes(x=k, y=mse_test_all[,3], color="WLLR")) +
labs(x="k", y="MSE", title="MSE on Test Data", color="Model") +
theme(plot.title=element_text(hjust=0.5)) +
scale_x_continuous(breaks = k_values) +
scale_color_manual(values = c("k-NN" = "#0000FF", "WKNN" = "#FF0000",
                              "WLLR" = "#00FF00")) +
theme(legend.position = c(1, 1), legend.justification = c(1, 1))
```



The test MSE for k-NN, weighted k-NN, and weighted local linear regression tells us the optimal model and k value based on test data performance. The weighted local linear regression consistently yields the lowest MSE across all k values greater or equal to 20. It leads us to believe it has an overall better fit. While k-NN and WkNN models demonstrate improved performance with an increase in k from 5 to 20, the reduction in MSE levels off for larger k values. Thus, the best performing procedure is WLLR, which achieves the most stable and lowest error, irrespective of the k value chosen. The minium MSE is achieved for $k = 100$ with an MSE test value of 1.0128524.