DSC1107_FA4-P2

John Benedict A. Monfero

March 5, 2025

Case Study: Apple Farming

- You own a square apple orchard, measuring 200 meters on each side. You have planted trees in a grid ten meters apart from each other. Last apple season, you measured the yield of each tree in your orchard (in average apples per week). You noticed that the yield of the different trees seems to be higher in some places of the orchard and lower in others, perhaps due to differences in sunlight and soil fertility across the orchard.

> Unbeknownst to you, the yield Y of the tree planted E_1 meters to the right and E_2 meters up from the bottom left-hand corner of the orchard has distribution $Y = f(E) + \epsilon$, where:

$$f(E) = 50 + 0.001(E_1)^2 + 0.001(E_2)^2 \text{ such that } \epsilon \sim N(0, \sigma^2) \text{ whereas } \sigma = 4$$

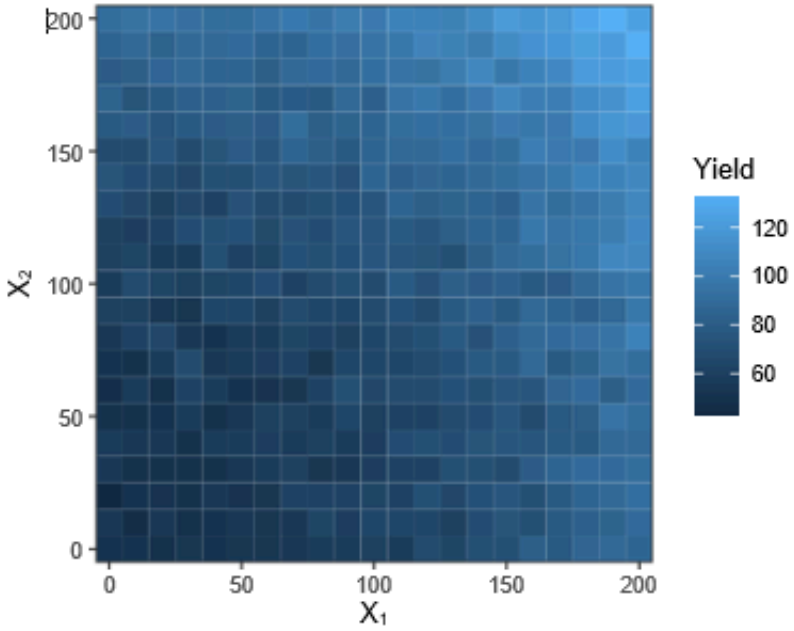


Figure 1: Apple tree yield for each 10m by 10m block of the orchard in a given year.

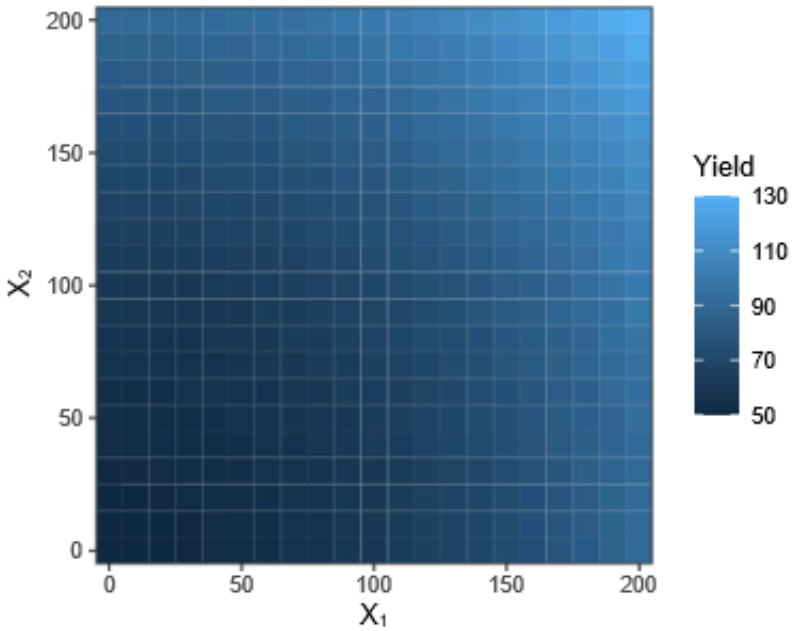


Figure 2: Underlying trend in apple yield for each 10m by 10m block of the orchard.

A Simple Rule to Predict this Season's Yeild

This apple season is right around the corner, and you'd like to predict the yield of each tree. You come up with perhaps the simplest possible prediction rule: predict this year's yield for any given tree based on last year's yield from that same tree. Without doing any programming, answer the following questions:

What is the training error of such a rule?

> The **training error** is the error when predicting the yield of each tree using the same data that was used to fit the model. Since the simplest prediction rule is to *use last year's yield to predict this year's yield*. Therefore, the training error will be based on the difference between the actual yield and the predicted yield (which is the same as supposed last year's yield).

Given the last year's yield model $Y = f(E) + \epsilon$ where:

$$f(E) = 50 + 0.001(E_1)^2 + 0.001(E_2)^2 \text{ such that } \epsilon \sim N(\mu = 0, \sigma^2 = 16)$$

> **The training error will be the variance of the error term** ϵ and since $\epsilon \sim N(\mu = 0, \sigma^2 = 16)$ that is ϵ indeed within Normally distributed with mean 0 and standard deviation 4. **Hence** $\text{training error } = \sigma^2$

What is the mean squared bias, mean variance, and expected test error of this prediction rule?

> **Mean Squared Bias** measures the error introduced by approximating a real-world problem, which may be complex, by a simplified model. However, since we are using last year's yield to predict this year's yield, the bias is essentially zero because we are not approximating but directly using the past value. Making $\texttt{MeanSquaredBias} = 0$

> On the other hand, **Mean Variance** measures how much the predictions for a given point vary between different realizations of the model. Since we are using the same yield from last year, the variance is the variance of the error term ϵ. Thus $\texttt{Variance} = \sigma^2 = 16$

> Finally, **Expected Test Error** is the expected value of the squared difference between the predicted and actual values on new data. It combines both the bias and the variance. Since the bias is zero, the expected test error is just the variance of the error term. Having:

$$\texttt{ExpectedTestError} = \texttt{Bias}^2 + \texttt{Variance} = 0 + 16 = 16$$

Why is this not the best possible prediction rule?

Using last year's yield to predict this year's yield is simple and intuitive, but it has some limitations and disregarding variability of the scenario throughout time as passes by include:

`Ignoring Environmental Changes`

`No Adaptation to Unpredicted Trends`

`Ignoring Individual Spatial Patterns`

`Statistical Oversimplified and Delimitations`

`Very Weak Model Complexity`

K-Nearest Neighbors (KNN) Regression (Conceptual)

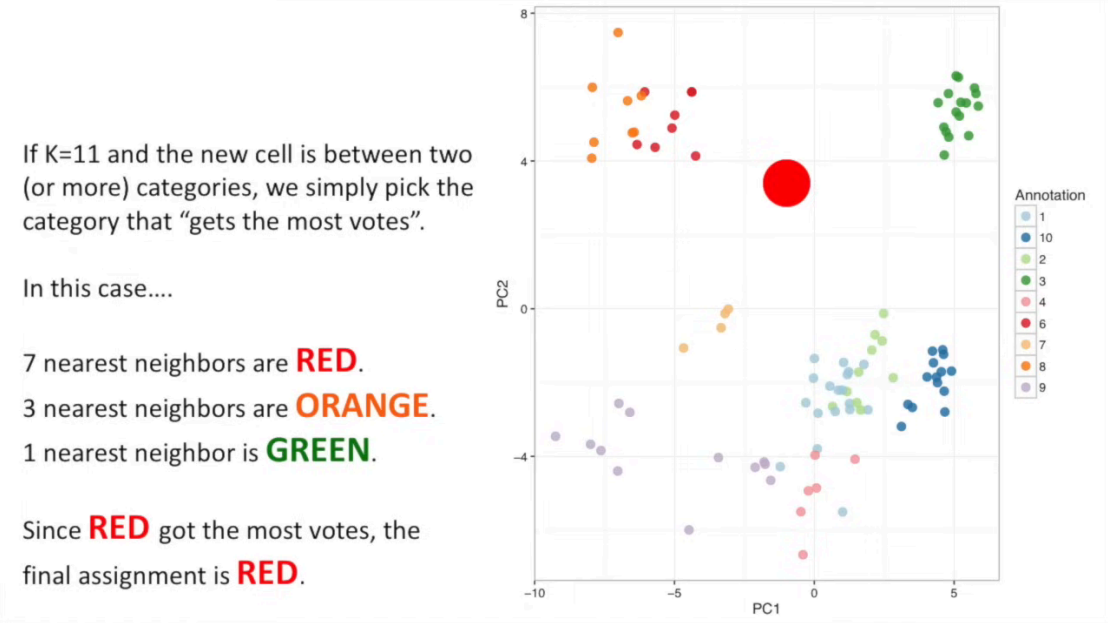
As a second attempt to predict a yield for each tree, you average together last year's yields of the K trees closest to it (including itself, and breaking ties randomly if necessary). So if you choose $K = 1$, you get back the simple rule from the previous section. This more general rule is called K-nearest neighbors (KNN) regression (see ISLR p. 105).

- **KNN** is not a parametric model like linear or logistic regression, so it is a little harder to pin down and narrow down its degrees of freedom.

1. What happens to the model complexity as K increases? Why?

Supposed there are dataset which already annotated their respective clusters (through PCA example), and we want to make classification for the new - unmarked - observation through which is the most rightful **nearest neighbor** shall it considers.

This is where K in “K-nearest neighbors” delegates the overall model complexity as new data introduces, for $K \in \mathbb{N}$ it defines how much nearest neighboring points shall be the comparison and basis to define the new data as this category. **Here a sample visual overview on how KNN works made by StatsQuest (2018)**



2. The degrees of freedom for KNN is sometimes considered n/K, where n is the training set size. Why might this be the case? [Hint: consider a situation where the data are clumped in groups of K.]

We know K is a parameter needed to make new observations to have predicted classifications based on how many K nearest neigbhors are. If the data naturally clumped (well distributed) in groups of K, then each clusters mentioned has low variability to even make distinction among their data points, which is enough to make the data points clustered as one huge circle point secured inside; recall that you have n amount of observation dataset size and whenever we divided by K clusters, **the model’s complexity or degrees of freedom is reduced and rather depends to the number of clusters, rather than the total number of data points.**

3. Conceptually, why might increasing K tend to improve the prediction rule? What does this have to do with the bias-variance tradeoff?

Consider n amount of datas to be trained, if K is not large enough, then the prediction rule may rather based on the outliers and noises of each clusters and referred as its nearest neighbors, which we do not want to promote underfitting the model’s desicion complexity. Thus increasing K large enough could optimize the prediction rule and would make the model be more accurate.

4. Conceptually, why might increasing K tend to worsen the prediction rule? What does this have to do with the bias-variance tradeoff?

Consider n amount of datas to be trained, if K is extremely and noticably large, then the prediction rule may be overwhelmed based on the sensitivity and apperance of other distinct clusters but considered as its nearest neighbors also, which we do not want to promote overfitting and extreme variance of the model’s desicion complexity. Thus increasing K unamaginable size a compared to n size, could hindered the prediction rule and would make the model less effective.

K-Nearest Neighbors (KNN) Regression (Simulation)

Now, we try KNN for several values of K. For each value of K, we use a numerical simulation to compute the bias and variance for every tree in the orchard. These results are contained in `training_results_summary` below.

```
training_results_summary <- readRDS("training_results_summary.rds")
glimpse(training_results_summary)
```

```
## Rows: 6,174
## Columns: 5
## $ K        <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 10, 8, 3, 9, 8, 9, 2, 3, 6, 7, …
## $ X1       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ X2       <dbl> 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 0, 10, 20, 30, 40, 50,…
## $ bias     <dbl> -0.25000000, 0.14000000, -0.52300000, 0.10900000, -0.56600000…
## $ variance <dbl> 16.20000, 12.20000, 20.40000, 15.60000, 21.40000, 15.90000, 1…
```

1. Create a new tibble called `overall_results` the contains the mean squared bias, mean variance, and expected test error for each value of K. This tibble should have four columns: `K`, `mean_sq_bias`, `mean_variance`, and `expected_test_error`.

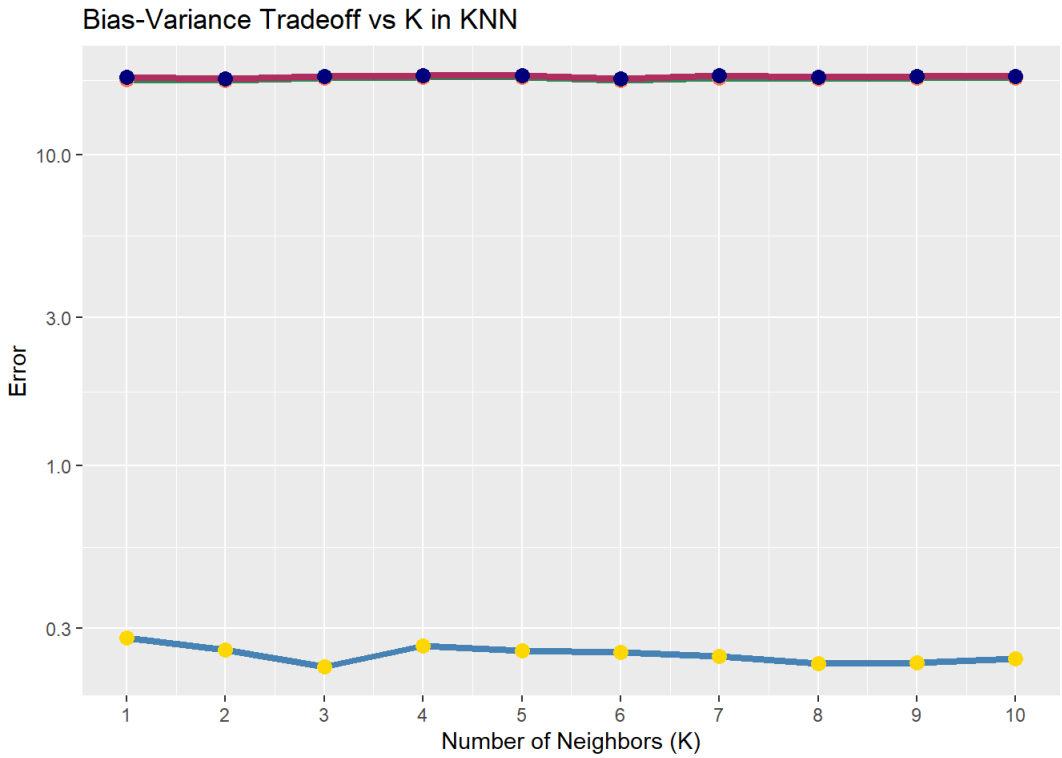
```
overall_results <- training_results_summary %>%
  group_by(K) %>%
  summarise(
    mean_sq_bias = mean(bias * bias),
    mean_variance = mean(variance),
    expected_test_error = mean_sq_bias + mean_variance
  )

overall_results
```

```
## # A tibble: 10 × 4
##        K mean_sq_bias mean_variance expected_test_error
##    <dbl>        <dbl>         <dbl>               <dbl>
##  1     1        0.280          17.5                17.8
##  2     2        0.256          17.4                17.6
##  3     3        0.227          17.7                17.9
##  4     4        0.264          17.7                18.0
##  5     5        0.255          17.8                18.0
##  6     6        0.252          17.3                17.6
##  7     7        0.244          17.7                18.0
##  8     8        0.232          17.6                17.8
##  9     9        0.232          17.6                17.9
## 10    10        0.240          17.6                17.9
```

2. Using `overall_results`, plot the mean squared bias, mean variance, and expected test error on the same axes as a function of K. Based on this plot, what is the optimal value of K?

```
overall_results %>%
  ggplot(mapping = aes(x = K)) +
  geom_line(aes(y = mean_sq_bias), col = "steelblue", lwd=1.5) +
  geom_point(aes(y = mean_sq_bias), col = "gold", size = 3) +
  geom_line(aes(y = mean_variance), col = "seagreen", lwd=1.5) +
  geom_point(aes(y = mean_variance), col = "coral", size = 3) +
  geom_line(aes(y = expected_test_error), col = "maroon", lwd=1.5) +
  geom_point(aes(y = expected_test_error), col = "navy", size = 3) +
  labs(
    title = "Bias-Variance Tradeoff vs K in KNN",
    x = "Number of Neighbors (K)",
    y = "Error",
    color = "Metric"
  ) +
  scale_x_continuous(breaks = seq(1, max(overall_results$K), by = 1)) +
  scale_y_log10()
```



3. We are used to the bias decreasing and the variance increasing when going from left to right in the plot. Here, the trend seems to be reversed. Why is this the case?

> The **`Expected Test Error`** significantly never changes much if as we minimizing the bias yet proportionally, we overely to variance of the observation. Recall that:

$$\texttt{Expected Test Error} = \texttt{mean(Bias}^2\texttt{)} + \texttt{mean(Variance)}$$

The only way to minimize **`Expected Test Error`** would be breaking the trend and reverse it such that:

$$\min[\texttt{Expected Test Error}] = \min[\texttt{mean(Bias}^2\texttt{)}] + \min[\texttt{mean(Variance)}]$$

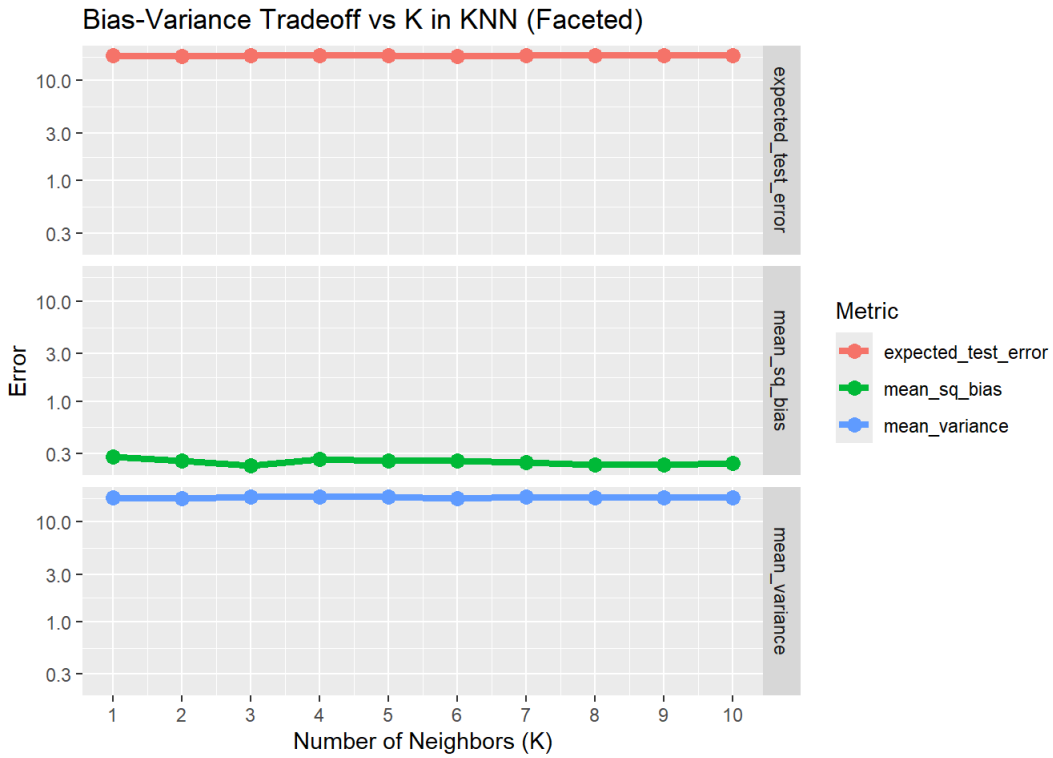
4. The mean squared bias has a strange bump between K = 1 and K = 5, increasing from K = 1 to K = 2 but then decreasing from K = 2 to K = 5. Why does this bump occur? [Hint: Think about the rectangular grid configuration of the trees. So for a given tree, the closest tree is itself, and then the next closest four trees are the ones that are one tree up, down, left, and right from it.]

> Consider the example above: At $K = 1$: The prediction for each tree depends only on itself

> The prediction truly starts incorporating the best one nearest neighbors, at $K = 2$ it is deciding which one of next closest four trees are the ones that are one tree up, down, left, and right from it.

> Finally, $K \in \{3, 4, 5\}$, the predictions become more stable and rely to some variability of the trees. Larger neighborhoods smooth out enough the influence of individual noisy points, leading to a decrease in bias and complexity of the model truly enters the game.

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



5. Based on the information in training_results_summary, which tree and which value of K gives the overall highest absolute bias? Does the sign of the bias make sense? Why do this particular tree and this particular value of K give us the largest absolute bias?

To find the tree and K with the highest absolute bias:

```
training_results_summary %>%
  filter(abs(bias) == max(abs(bias)))
```

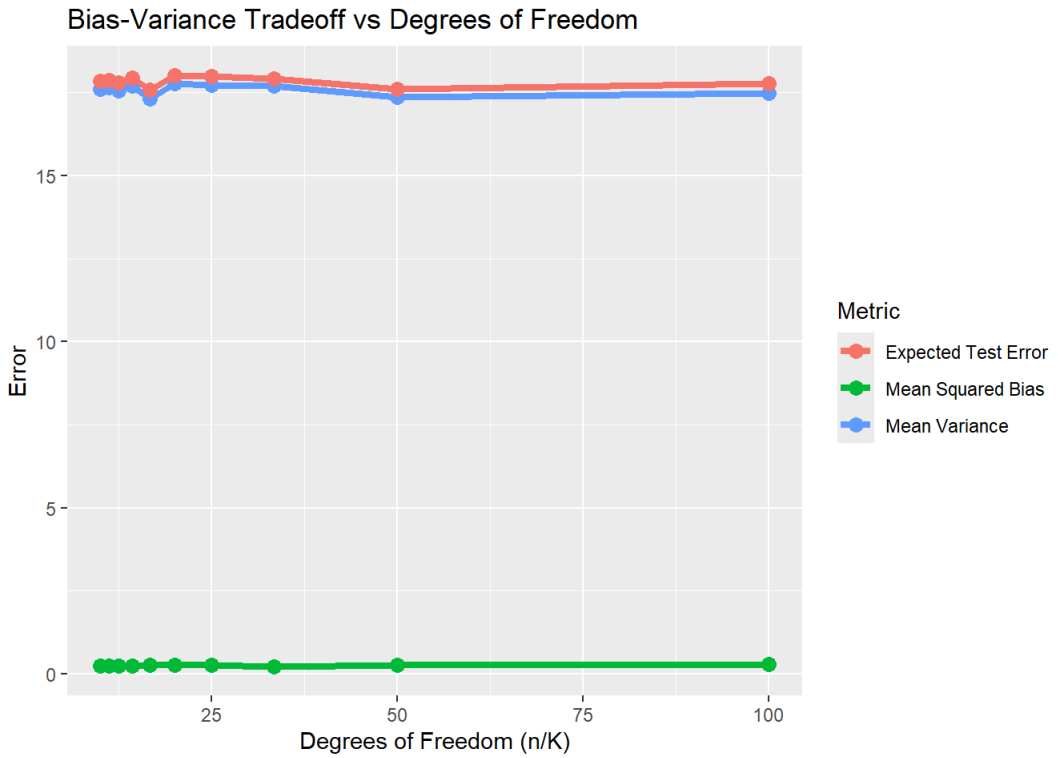
```
## # A tibble: 1 × 5
##       K    X1    X2  bias variance
##   <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1     1     0    70 -2.06     13.9
```

- The overall highest absolute bias happens at $\min(K)$
- A *negative bias* indicates predictions are consistently below true values.
- A *positive bias* suggests that predictions consistently overshoot true values.
- A small K could amplify the effect because its predictions rely only on very few neighbors, leading to localized dependencies but globally inaccuracies.

6. Redo the bias-variance plot from part 2, this time putting df = n/K on the x-axis. What do we notice about the variance as a function of df?

```
n <- max(overall_results$K) * 10  # Assume n (training set size)
overall_results <- overall_results %>%
  mutate(df = n / K)  # Degrees of freedom

# Plot bias-variance with df on x-axis
ggplot(overall_results, aes(x = df)) +
  geom_line(aes(y = mean_sq_bias, color = "Mean Squared Bias"), size = 1.5) +
  geom_point(aes(y = mean_sq_bias, color = "Mean Squared Bias"), size = 3) +
  geom_line(aes(y = mean_variance, color = "Mean Variance"), size = 1.5) +
  geom_point(aes(y = mean_variance, color = "Mean Variance"), size = 3) +
  geom_line(aes(y = expected_test_error, color = "Expected Test Error"), size = 1.5) +
  geom_point(aes(y = expected_test_error, color = "Expected Test Error"), size = 3) +
  labs(
    title = "Bias-Variance Tradeoff vs Degrees of Freedom",
    x = "Degrees of Freedom (n/K)",
    y = "Error",
    color = "Metric"
  )
```



`As df` $\to \infty$ `it means that` $K \to 0^+$ and variance tends to increase

7. Derive a formula for the KNN mean variance. [Hint: First, write down an expression for the KNN prediction for a given tree. Then, compute the variance of this quantity using the fact that the variance of the average of N independent random variables each with variance s2 is s2/N . Finally, compute the mean variance by averaging over trees.]

`KNN Prediction Formula would be` $\hat{y}(x) := \frac{1}{K}\sum_{i=1}^{K}(y_i)$

Where y_i represents the response value for the i^{th} nearest neighbor.

Assuming that the y_i values are independent and identically distributed (by definition of `var()`) with variance σ^2, the variance of the KNN prediction $\hat{y}(x)$ is computed as:

$$\text{Var}(\hat{y}(x)) = \text{Var}\left(\frac{1}{K}\sum_{i=1}^{K} y_i\right)$$

Recall that using the formula - above - for the variance is also the average of K independent random variables: $E(X) = \mu$

To compute the mean variance, average the variance of the KNN prediction over all query points (trees):

$$\text{Mean Variance} = \frac{\sigma^2}{K}$$

8. Create a plot like that in part 6, but with the mean variance formula from part 7 superimposed as a dashed curve. Do these two variance curves match?

```
sigma_squared <- mean(training_results_summary$variance) * 10  # Estimate sigma^2 from the data
overall_results <- overall_results %>%
  mutate(theoretical_variance = sigma_squared / K)  # Add the theoretical variance
```

```
ggplot(overall_results, aes(x = df)) +
  geom_line(aes(y = mean_variance, color = "Mean Variance (Empirical)"), size = 1.5) +
  geom_point(aes(y = mean_variance, color = "Mean Variance (Empirical)"), size = 3) +
  geom_line(aes(y = theoretical_variance, color = "Mean Variance (Theoretical)"),
            linetype = "dashed", size = 1.2) +  # Add dashed theoretical curve
  labs(
    title = "Variance vs Degrees of Freedom with Theoretical Curve",
    x = "Degrees of Freedom (n/K)",
    y = "Variance",
    color = "Metric"
  ) +
  scale_x_continuous(breaks = seq(0, max(overall_results$df), by = 10)) +
  scale_y_log10()
```