Final Project

December 12, 2023

We decided to split our data and test each method into two sections: Simulated data, and real-world data. This is so we could see how it would work with both datasets, specifically when adding known, intentional outliers versus real-world but questionable outliers. We first run on all the methods we described, and then use MSE and R-Squared to compare performances of each method versus the controls of running on Mahalanobis Distance and also not removing any outliers at all.

Simulated Data

Simple Polynomial Simulation

For our first simulation, I generated a simple quadratic polynomial based off of this equation:

$$y = 5x^2 + 2x + 4$$

```
# Generate simulated polynomial data
x1poly = runif(50, min = -1, max = 1)
ypoly = 5*x1poly^2 + 2*x1poly + 4 + rnorm(50, mean = 0, sd = 1)

# Combine the data into a dataframe
poly_df = data.frame(x1poly, ypoly)
# Split into training and testing sets
poly_train = poly_df[1:40, ]
poly_test = poly_df[41:50, ]
```

We also want to add our explicit outliers - these will be points that are at least 3 standard deviations away from the mean. Note that unlike real data, we KNOW these are outliers, and this will be of note later on. We also added 6 points to keep a ratio of around 10% outliers to total points.

```
# Add explicit outliers
coefficients_to_use = rnorm(6, mean = 3, sd = 1)
poly_outliers = data.frame(x1poly = c(-0.4, -0.2, 0, 0.2, 0.4, 0.6), ypoly = c(mean(poly_train$ypoly) +
poly_outliers
```

```
##   x1poly      ypoly
## 1   -0.4 12.1449232
## 2   -0.2  0.7104625
## 3    0.0 -0.1584343
## 4    0.2 11.7204342
## 5    0.4 10.4224337
## 6    0.6 10.4632045
```

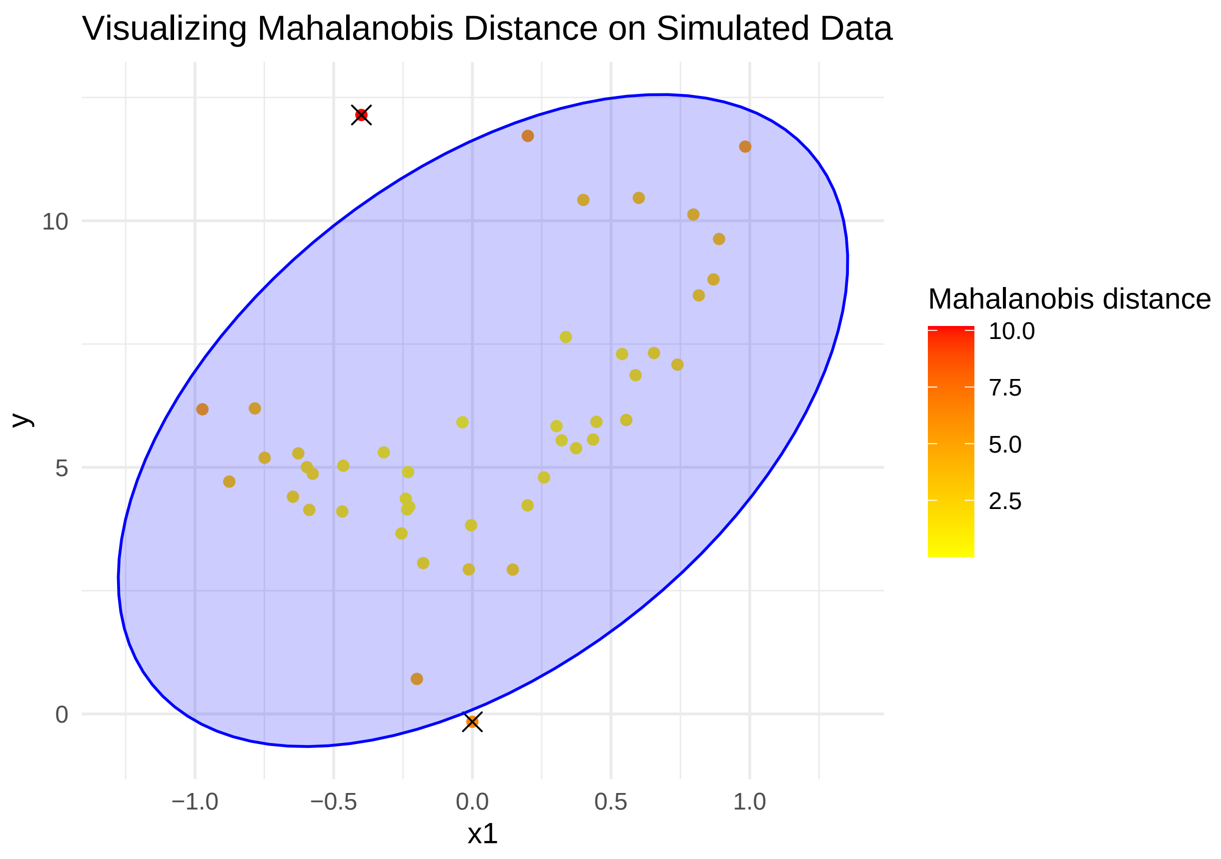
```
poly_train = rbind(poly_train, poly_outliers)
```

Now let's calculate the Mahalanobis distance for each point in the training set based off of the 95th percentile, and plot the data with the Mahalanobis distance ellipses. Note that for each plot we make, an x is marked on points considered outliers by the respective method.

```r
mana_dst = mahalanobis(poly_train, colMeans(poly_train), cov(poly_train))
mana_cutoff = qchisq(0.95, df=ncol(poly_train))

# Get outliers with the cutoff
mana_outliers = poly_train[mana_dst > mana_cutoff, ]
mana_outliers
```

```
##    x1poly      ypoly
## 41   -0.4 12.1449232
## 43    0.0 -0.1584343
```

```r
poly_ellipse = as.data.frame(ellipse(cor(poly_train), scale=c(sd(poly_train$x1poly), sd(poly_train$ypol
ggplot(poly_train, aes(x = x1poly, y = ypoly)) +
  geom_point(aes(color=mana_dst)) +
  geom_point(data = mana_outliers, shape = 4, size=3) +
  geom_polygon(data = poly_ellipse, alpha = 0.2, fill = "blue", color = "blue") +
  labs(title = "Visualizing Mahalanobis Distance on Simulated Data", x = "x1", y = "y", color="Mahalano
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



We can see an ellipse that encompasses the 95th percentile of spread from all the points. We can see that Mahalanobis distance was pretty conservative on labeling outliers, as it only labeled 2 points as outliers.

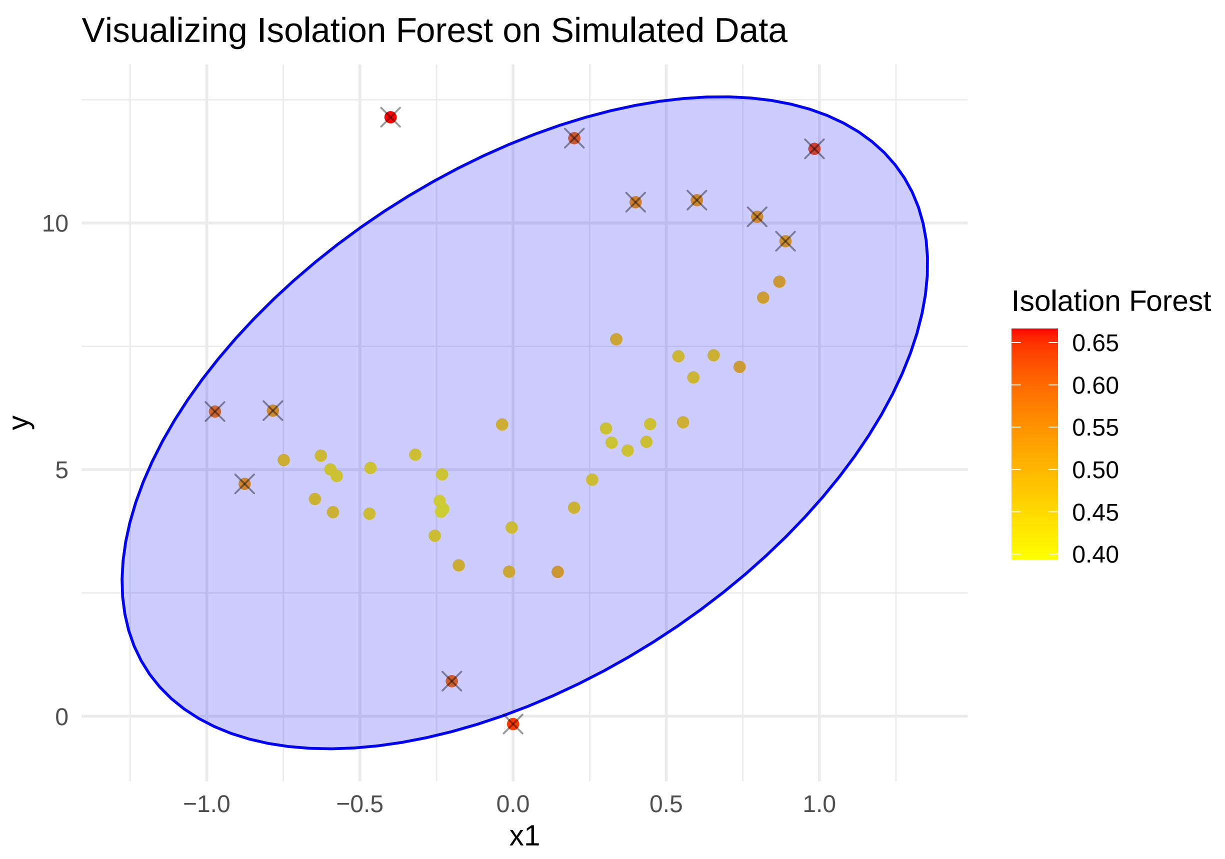
Now we are going to use Isolation Forest to detect outliers in the simulated data. The threshold of 0.5 is used because if it is above 0.5 probability of being an outlier we will consider it as such. We decided to do 500 for ntrees because our observations saw that this number of trees seemed to give good performance in terms of detecting actual outliers.

```r
# Use Isolation Forest to detect outliers
if_model = isolation.forest(poly_train, ntree = 500, nthreads = 1)
if_outliers_pred = predict(if_model, poly_train)
if_outliers = poly_train[if_outliers_pred > .5, ]
if_outliers
```

```
##        x1poly      ypoly
## 6   0.7967794 10.1265251
## 7   0.8893505  9.6306352
## 10 -0.8764275  4.7107110
## 18  0.9838122 11.5040199
## 27 -0.9732193  6.1773143
## 38 -0.7841127  6.1956779
## 41 -0.4000000 12.1449232
## 42 -0.2000000  0.7104625
## 43  0.0000000 -0.1584343
## 44  0.2000000 11.7204342
## 45  0.4000000 10.4224337
## 46  0.6000000 10.4632045
```

Let's plot with the Isolation Forest outliers colored.

```r
# Plot the data with the outliers
ggplot(poly_train, aes(x = x1poly, y = ypoly)) +
  geom_point(aes(color=if_outliers_pred)) +
  geom_polygon(data = poly_ellipse, alpha = 0.2, fill = "blue", color = "blue") +
  geom_point(data = if_outliers, shape = 4, size=3, alpha=0.4) +
  labs(title = "Visualizing Isolation Forest on Simulated Data", x = "x1", y = "y", color="Isolation Fo
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



Here we can see that Isolation Forest was much more aggressive in labeling outliers, as it labeled 12 points as outliers. The measure seems to be on the higher side for a lot of points even if they may be on the distribution.

Now, we are going to use local outlier factor to detect outliers in the simulated data. I use the number 1.5 as the threshold as anything above 1 could be considered an outlier.

```
# Use local outlier factor to detect outliers
lof = lof(poly_train, k = 5)
```

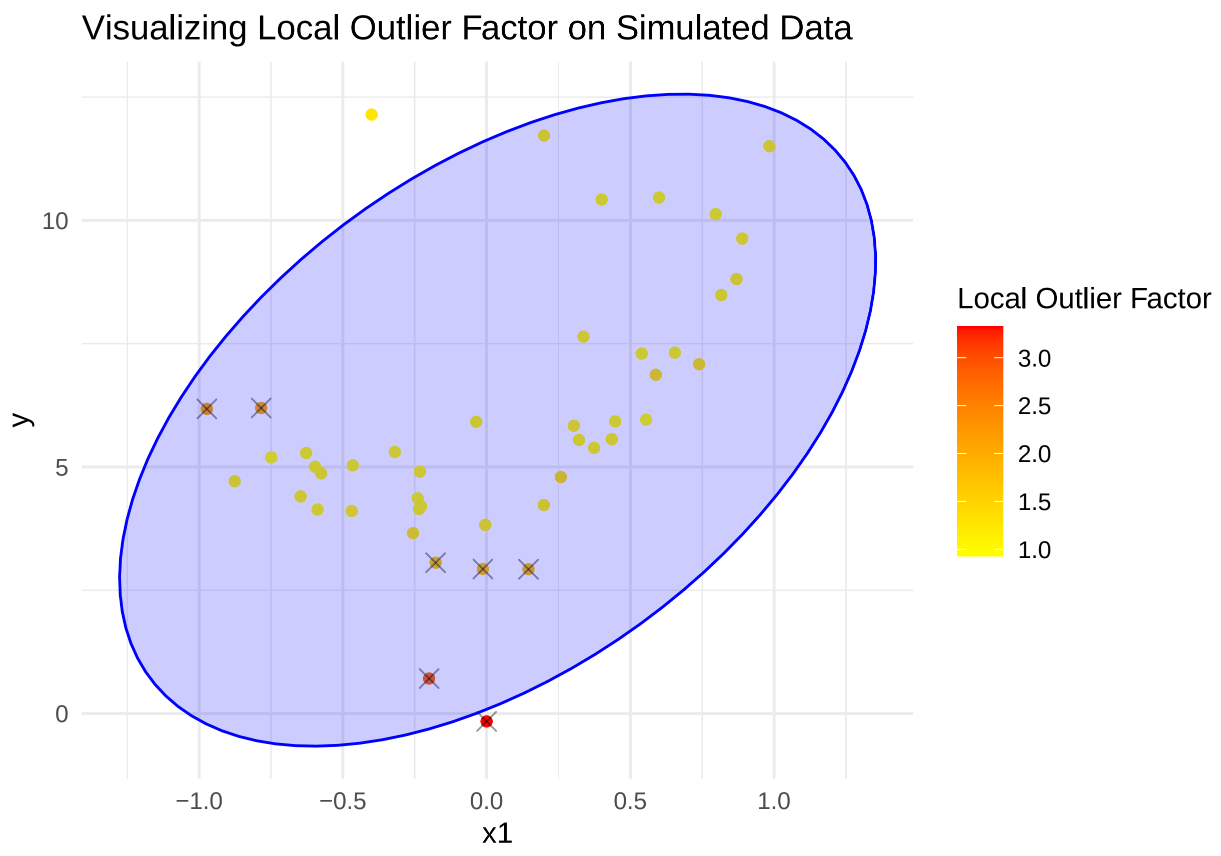
```
## Warning in lof(poly_train, k = 5): lof: k is now deprecated. use minPts = 6
## instead .
```

```
lof_outliers = poly_train[lof > 1.5, ]
lof_outliers
```

```
##         x1poly      ypoly
## 3   0.14570673  2.9268133
## 27 -0.97321933  6.1773143
## 33 -0.01291739  2.9308649
## 38 -0.78411275  6.1956779
## 40 -0.17745114  3.0592690
## 42 -0.20000000  0.7104625
## 43  0.00000000 -0.1584343
```

Let’s plot with the local outliers colored..

```
# Plot the data with the outliers
ggplot(poly_train, aes(x = x1poly, y = ypoly)) +
  geom_point(aes(color=lof)) +
  geom_point(data = lof_outliers, shape = 4, size=3, alpha=0.4) +
  geom_polygon(data = poly_ellipse, alpha = 0.2, fill = "blue", color = "blue") +
  labs(title = "Visualizing Local Outlier Factor on Simulated Data", x = "x1", y = "y", color="Local Ou
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



Local outlier factor seems to be somewhat conservative in labelling points, but has a large quantity of points that seem on the larger side.

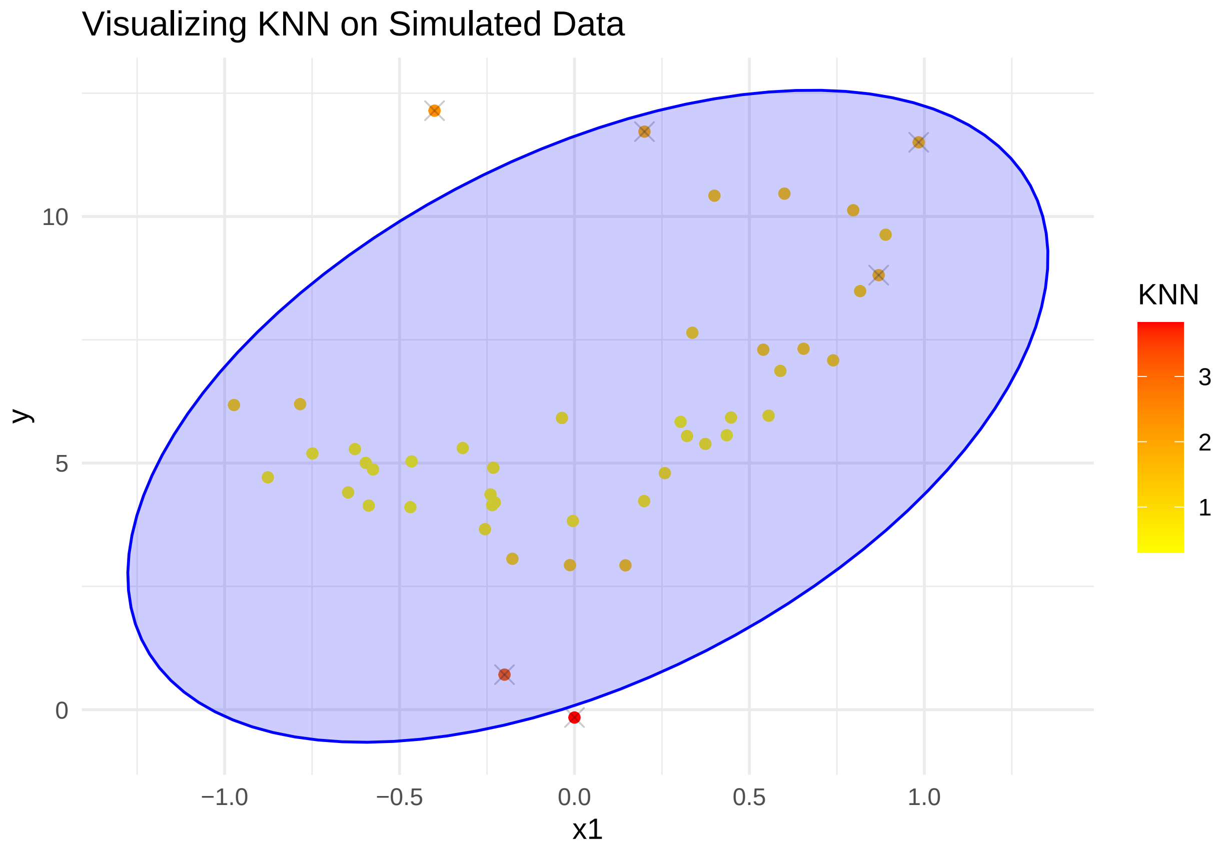
Finally, we'll use KNN to detect outliers in the simulated data. We use 5 as we saw K=5 as a relatively reliable number to use, and used 1.5 as the threshold for this example as a point at least 1.5 units away from the other points would be reasonable to designate as an outlier.

```
# Use KNN to detect outliers
knn_model = get.knn(poly_train, k = 5)
knn_dist = knn_model$nn.dist[,5]
knn_outliers = poly_train[knn_dist > 1.5, ]
knn_outliers
```

```
##        x1poly      ypoly
## 18  0.9838122 11.5040199
## 21  0.8694105  8.8106985
## 41 -0.4000000 12.1449232
## 42 -0.2000000  0.7104625
## 43  0.0000000 -0.1584343
## 44  0.2000000 11.7204342
```

Let's plot the data with the KNN outliers colored.

```
# Plot the data with the outliers
ggplot(poly_train, aes(x = x1poly, y = ypoly)) +
  geom_point(aes(color=knn_dist)) +
  geom_point(data = knn_outliers, shape = 4, size=3, alpha=0.2) +
  geom_polygon(data = poly_ellipse, alpha = 0.2, fill = "blue", color = "blue") +
  labs(title = "Visualizing KNN on Simulated Data", x = "x1", y = "y", color="KNN") +
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```

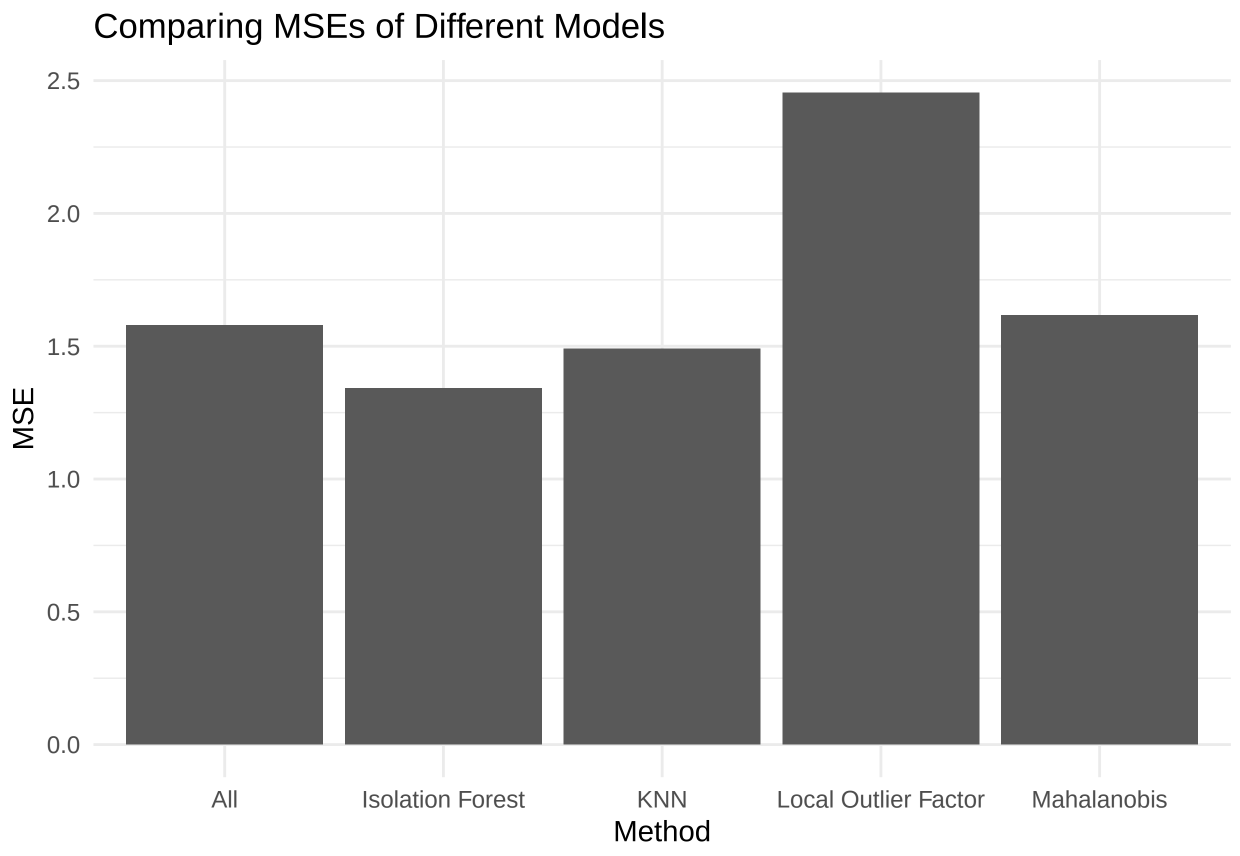


KNN Seems to be very conservative in labelling points compared to the other methods, as it only labels 4 points as outliers.

Let's now generate a mutli-linear regression model using this trainng data, and using it to predict the test data to calculate the MSE's for each method. We first generate a model for a quadratic function without removing any outliers, then create regression models for each method removing its respective determined outliers.

```
# Get the training data without the Mahalanobis outliers by removing points that are in mana_outliers
poly_train_mana = poly_train[!poly_train$x1poly %in% mana_outliers$x1poly, ]
# Get data without the IF outliers
poly_train_if = poly_train[!poly_train$x1poly %in% if_outliers$x1poly, ]
# Get data without the LOF outliers
poly_train_lof = poly_train[!poly_train$x1poly %in% lof_outliers$x1poly, ]
# Get data without the KNN outliers
poly_train_knn = poly_train[!poly_train$x1poly %in% knn_outliers$x1poly, ]
# Generate models for all
poly_lm = lm(ypoly ~ x1poly+I(x1poly^2), data = poly_train)
poly_lm_mana = lm(ypoly ~ x1poly+I(x1poly^2), data = poly_train_mana)
```

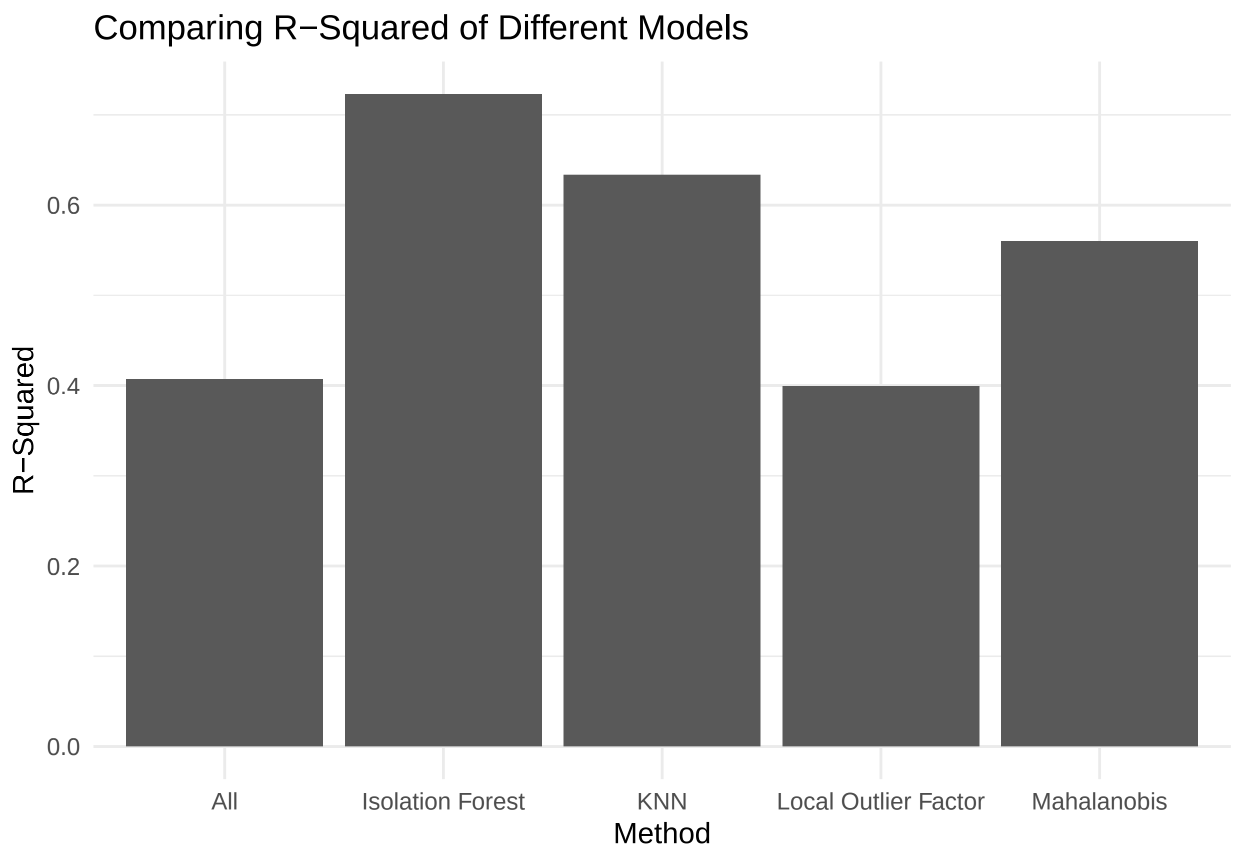
```
poly_lm_if = lm(ypoly ~ x1poly+I(x1poly^2), data = poly_train_if)
poly_lm_lof = lm(ypoly ~ x1poly+I(x1poly^2), data = poly_train_lof)
poly_train_knn = lm(ypoly ~ x1poly+I(x1poly^2), data = poly_train_knn)
# Calculate MSE
mana_mse = mean((poly_test$ypoly - predict(poly_lm_mana, poly_test))^2)
if_mse = mean((poly_test$ypoly - predict(poly_lm_if, poly_test))^2)
lof_mse = mean((poly_test$ypoly - predict(poly_lm_lof, poly_test))^2)
knn_mse = mean((poly_test$ypoly - predict(poly_train_knn, poly_test))^2)
all_mse = mean((poly_test$ypoly - predict(poly_lm, poly_test))^2)

# Compare the MSEs by plotting them
mse_df = data.frame(mse = c(all_mse, mana_mse, if_mse, lof_mse, knn_mse),
                    method = c("All", "Mahalanobis", "Isolation Forest", "Local Outlier Factor", "KNN")
# Shrink y scale to make the plot more readable
ggplot(mse_df, aes(x = method, y = mse)) +
  geom_bar(stat = "identity") +
  labs(title = "Comparing MSEs of Different Models", x = "Method", y = "MSE") +
  theme_minimal()
```



Let's also compare the R-Squared values of the methods.

```
# Calculate R-Squared
all_r2 = summary(poly_lm)$r.squared
mana_r2 = summary(poly_lm_mana)$r.squared
if_r2 = summary(poly_lm_if)$r.squared
lof_r2 = summary(poly_lm_lof)$r.squared
```

```r
knn_r2 = summary(poly_train_knn)$r.squared

# Compare the R-Squared values by plotting them
r2_df = data.frame(r2 = c(all_r2, mana_r2, if_r2, lof_r2, knn_r2),
                   method = c("All", "Mahalanobis", "Isolation Forest", "Local Outlier Factor", "KNN"))
# Shrink y scale to make the plot more readable
ggplot(r2_df, aes(x = method, y = r2)) +
  geom_bar(stat = "identity") +
  labs(title = "Comparing R-Squared of Different Models", x = "Method", y = "R-Squared") +
  theme_minimal()
```



We can see that with MSE, Mahalanobis does pretty well (for this run), but with R-Sqaured Isolation Forest performs pretty well, better than Mahalanobis. In this run interesetingly KNN seems to have performed pretty well in getting the obvious outliers, so it seems as the conservative approach worked pretty well. However, with the real-world data it seems to not perform really well, which we will talk about in that section. Unfortunately in this case it seems as LOF don't perform well with MSE or R-Squared.

Complex Simulation

Now we will do the same procedures with a multi-variable polynomial regression model. We will generate the data, add outliers, and then fit the model.

```r
# Same procedure, except with a mutlivariable underlying
set.seed(1)
x1complex = runif(50, min = -1, max = 1)
x2complex = runif(50, min = -1, max = 1)
```

```r
ycomplex = 2*x1complex^2 + 3*x1complex + 4 + 2*x2complex^2 + 3*x2complex + 4 + rnorm(50, mean = 0, sd =
# Get mean of y's with only x1 complex
x1mean = mean(ycomplex[x1complex > 0])
x2mean = mean(ycomplex[x2complex > 0])

# Make into data frame
complex_data = data.frame(x1complex, x2complex, ycomplex)
# Split into train and test
complex_train = complex_data[1:40, ]
complex_test = complex_data[41:50, ]
```

Adding explicit outliers, with outliers for only x1 and then only x2, and then points that are the same for x1 and x2 that should be outliers for both, as we add around 3 standard deviations to the respective mean (x1's mean, x2's mean, and y's overall mean respectively):

```r
# Create explicit outliers as similar to before
coefficients_to_use = rnorm(7, mean=3, sd=1)
# Create outliers for x1
outlierscomplexx1 = c(-0.7, -0.3, -0.1, 0.1)
# Create x values for x2 that are different points
outlierscomplexx2 = c(-0.9, -0.2, 0, 0.6)
# x values that are the same points for x1 and x2
outlierscomplexx1x2 = c(-0.8, -0.4, 0.2)
# Create y outlier values

outlierscomplexy = c(x1mean - coefficients_to_use[1]*sd(ycomplex[x1complex > 0]),
                     x1mean + coefficients_to_use[2]*sd(ycomplex[x1complex > 0]),
                     x2mean + coefficients_to_use[3]*sd(ycomplex[x2complex > 0]),
                     x2mean - coefficients_to_use[4]*sd(ycomplex[x2complex > 0]),
                     mean(ycomplex) + coefficients_to_use[5]*sd(ycomplex),
                     mean(ycomplex) - coefficients_to_use[6]*sd(ycomplex),
                     mean(ycomplex) + coefficients_to_use[7]*sd(ycomplex))

# Add the outliers to the data
x1complex = c(complex_train$x1complex, outlierscomplexx1)
x1complex = c(x1complex, outlierscomplexx1x2)
x2complex = c(complex_train$x2complex, outlierscomplexx2)
x2complex = c(x2complex, outlierscomplexx1x2)
ycomplex = c(complex_train$ycomplex, outlierscomplexy)

# Merge into train data
complex_train = data.frame(x1complex, x2complex, ycomplex)
complex_train
```

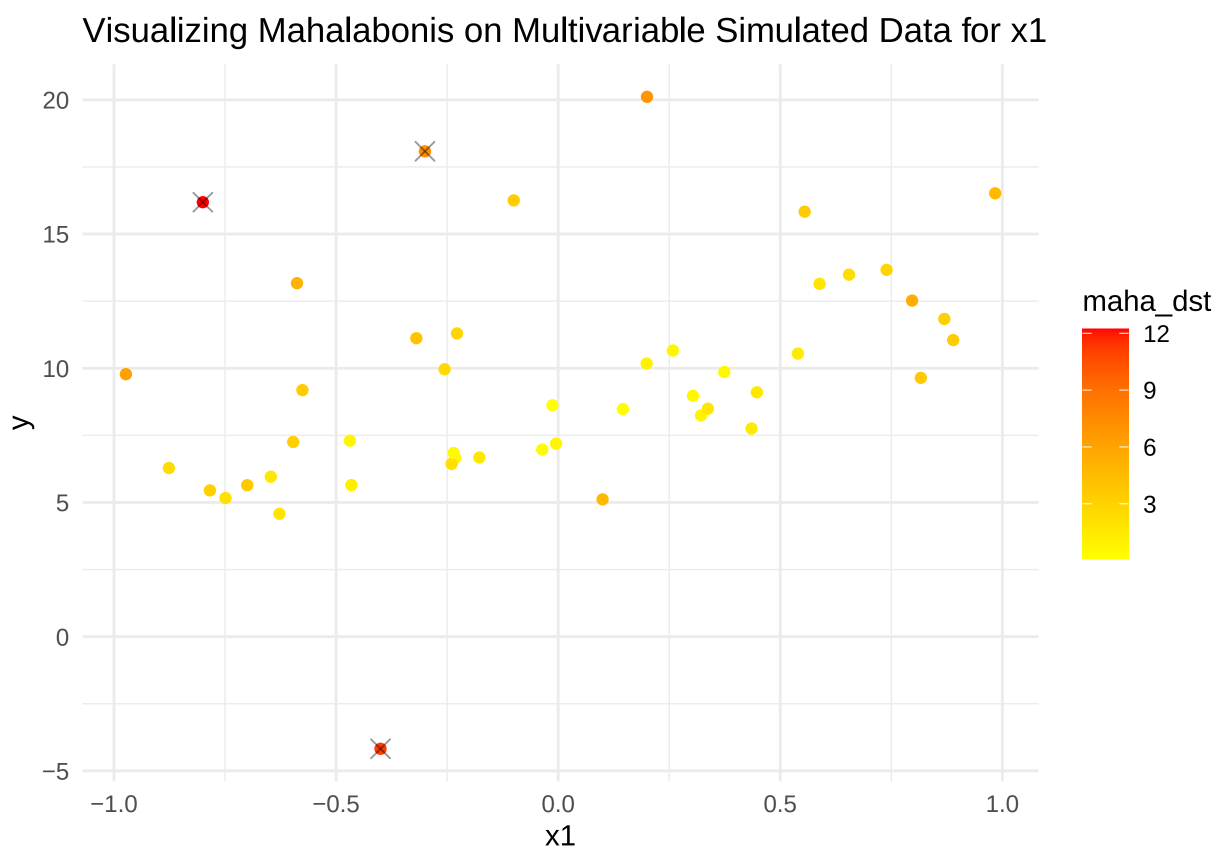
```
##       x1complex   x2complex  ycomplex
## 1  -0.468982674 -0.04476076  7.300772
## 2  -0.255752201  0.72241895  9.962571
## 3   0.145706727 -0.12380579  8.479939
## 4   0.816415580 -0.51040545  9.642764
## 5  -0.596636138 -0.85864191  7.253671
## 6   0.796779370 -0.80106768 12.520669
## 7   0.889350537 -0.36745659 11.050398
## 8   0.321595585  0.03726853  8.242083
## 9   0.258228088  0.32401015 10.659763
## 10 -0.876427459 -0.18633963  6.282339
## 11 -0.588050850  0.82575185 13.170061
## 12 -0.646886495 -0.41279325  5.959442
## 13  0.374045693 -0.08186855  9.859496
## 14 -0.231792564 -0.33521065  6.659180
## 15  0.539682840  0.30174093 10.545608
## 16 -0.004601516 -0.48396644  7.191578
## 17  0.435237017 -0.04290950  7.754569
## 18  0.983812190  0.53262134 16.517999
## 19 -0.239929641 -0.83150617  6.436883
## 20  0.554890443  0.75064266 15.831947
## 21  0.869410462 -0.32185412 11.837108
## 22 -0.575714957  0.67888070  9.184204
## 23  0.303347532 -0.30663302  8.972957
## 24 -0.748889808 -0.33245014  5.164601
## 25 -0.465558663 -0.04729751  5.645762
## 26 -0.227771815  0.78439667 11.295637
## 27 -0.973219334  0.72867894  9.779345
## 28 -0.235224086 -0.22002091  6.842850
## 29  0.739381691  0.55464140 13.665035
## 30 -0.319302007  0.92123599 11.117540
## 31 -0.035839769 -0.13068103  6.968493
## 32  0.199131651  0.42502936 10.177911
## 33 -0.012917386 -0.20001126  8.619644
## 34 -0.627564797 -0.34929570  4.577542
## 35  0.654746637  0.51417430 13.486846
## 36  0.336933476 -0.59461549  8.494088
## 37  0.588479721  0.42224244 13.144460
## 38 -0.784112748 -0.75661616  5.448231
## 39  0.447421892 -0.50902297  9.103797
## 40 -0.177451141 -0.71339124  6.675404
## 41 -0.700000000 -0.90000000  5.641235
## 42 -0.300000000 -0.20000000 18.083017
## 43 -0.100000000  0.00000000 16.253890
## 44  0.100000000  0.60000000  5.113406
## 45 -0.800000000 -0.80000000 16.186199
## 46 -0.400000000 -0.40000000 -4.179040
## 47  0.200000000  0.20000000 20.112512
```

Now we will calculate the mahanalobis distance for each point to determine outliers. Unless otherwise noted the hyperparameters will be the same as the simple model.

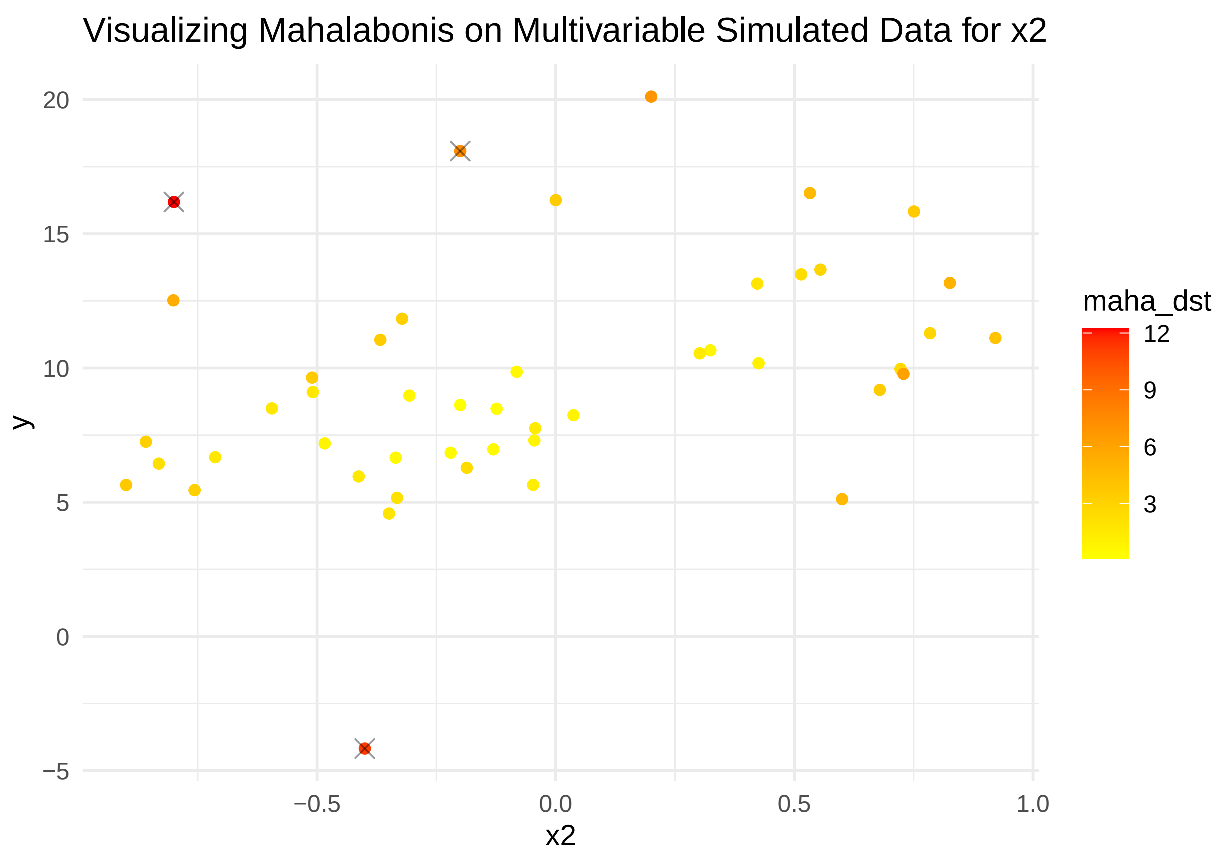
```r
maha_dst = mahalanobis(complex_train, colMeans(complex_train), cov(complex_train))
maha_cutoff = quantile(maha_dst, 0.95)
maha_outliers = which(maha_dst > maha_cutoff)
```

Becuase this is 3-Dimensional, we will just plot y on x1 and x2 individually and the respective Mahalanobis distances.

```r
# Plot the data, x1 first
ggplot(complex_train, aes(x = x1complex, y = ycomplex)) +
  geom_point(aes(col=maha_dst)) +
  geom_point(data = complex_train[maha_outliers, ], shape = 4, size=3, alpha=0.4) +
  labs(title = "Visualizing Mahalabonis on Multivariable Simulated Data for x1", x = "x1", y = "y") +
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



```r
# Plot the data, x2 now
ggplot(complex_train, aes(x = x2complex, y = ycomplex)) +
  geom_point(aes(col=maha_dst)) +
  geom_point(data = complex_train[maha_outliers, ], shape = 4, size=3, alpha=0.4) +
  labs(title = "Visualizing Mahalabonis on Multivariable Simulated Data for x2", x = "x2", y = "y") +
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



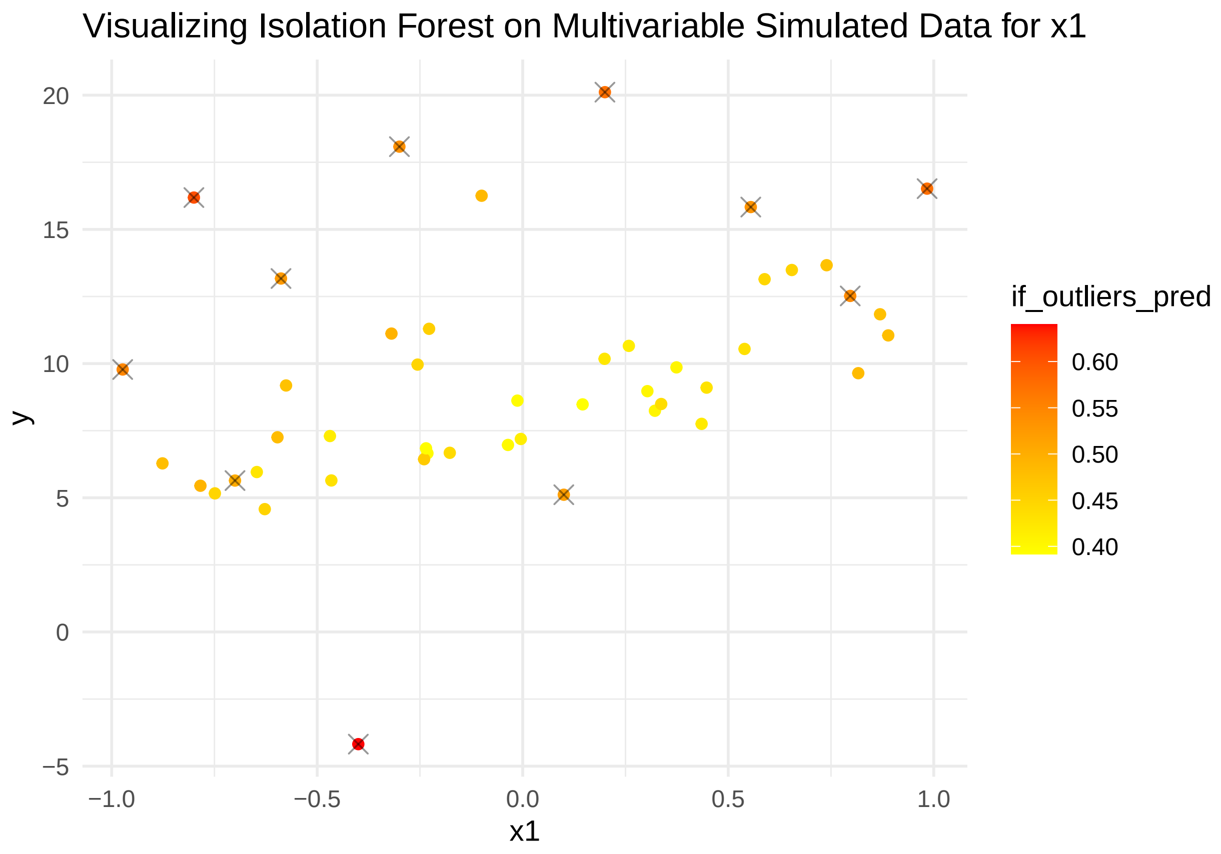
We can see that the Mahalanobis distance is not as agressive as it was in the simple model.

Using Isolation Forest

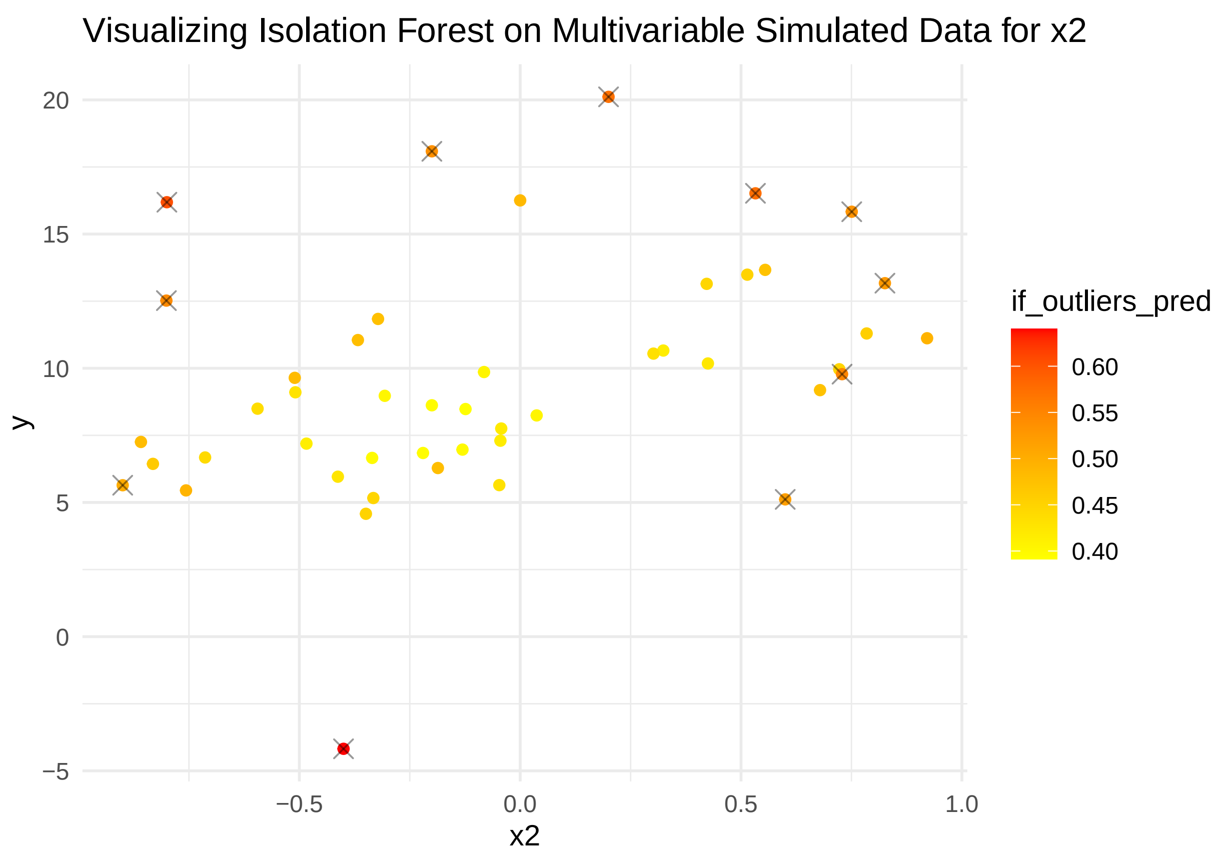
```
if_model = isolation.forest(complex_train, ntree = 500, ndim = 2)
if_outliers_pred = predict(if_model, complex_train)
if_outliers = complex_train[if_outliers_pred > .5, ]
if_outliers
```

```
##     x1complex  x2complex  ycomplex
## 6   0.7967794 -0.8010677 12.520669
## 11 -0.5880509  0.8257518 13.170061
## 18  0.9838122  0.5326213 16.517999
## 20  0.5548904  0.7506427 15.831947
## 27 -0.9732193  0.7286789  9.779345
## 41 -0.7000000 -0.9000000  5.641235
## 42 -0.3000000 -0.2000000 18.083017
## 44  0.1000000  0.6000000  5.113406
## 45 -0.8000000 -0.8000000 16.186199
## 46 -0.4000000 -0.4000000 -4.179040
## 47  0.2000000  0.2000000 20.112512
```

```
# Plot the data, x1 first
ggplot(complex_train, aes(x = x1complex, y = ycomplex)) +
  geom_point(aes(col=if_outliers_pred)) +
  labs(title = "Visualizing Isolation Forest on Multivariable Simulated Data for x1", x = "x1", y = "y")
  geom_point(data = if_outliers, shape = 4, size=3, alpha=0.4) +
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



```
# Plot the data, x2 now
ggplot(complex_train, aes(x = x2complex, y = ycomplex)) +
  geom_point(aes(col=if_outliers_pred)) +
  labs(title = "Visualizing Isolation Forest on Multivariable Simulated Data for x2", x = "x2", y = "y"
  geom_point(data = if_outliers, shape = 4, size=3, alpha=0.4) +
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



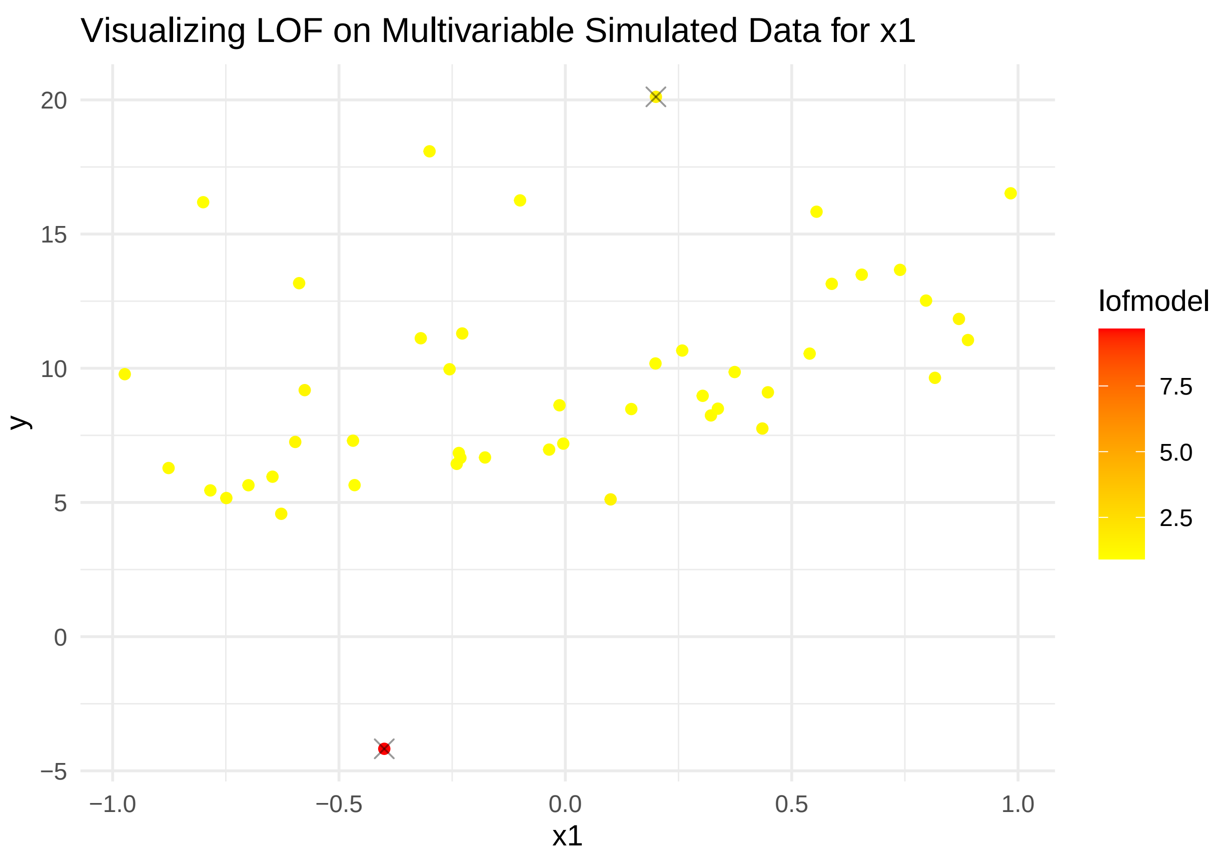
Isolation forest has marked the obvious outliers, but has also marked points (with our threshold) that seem to be within the distribution. It seems like points that are correlated in x2's dimension but are outliers in x1's are indeed marked as outliers.

Using local outlier factor:

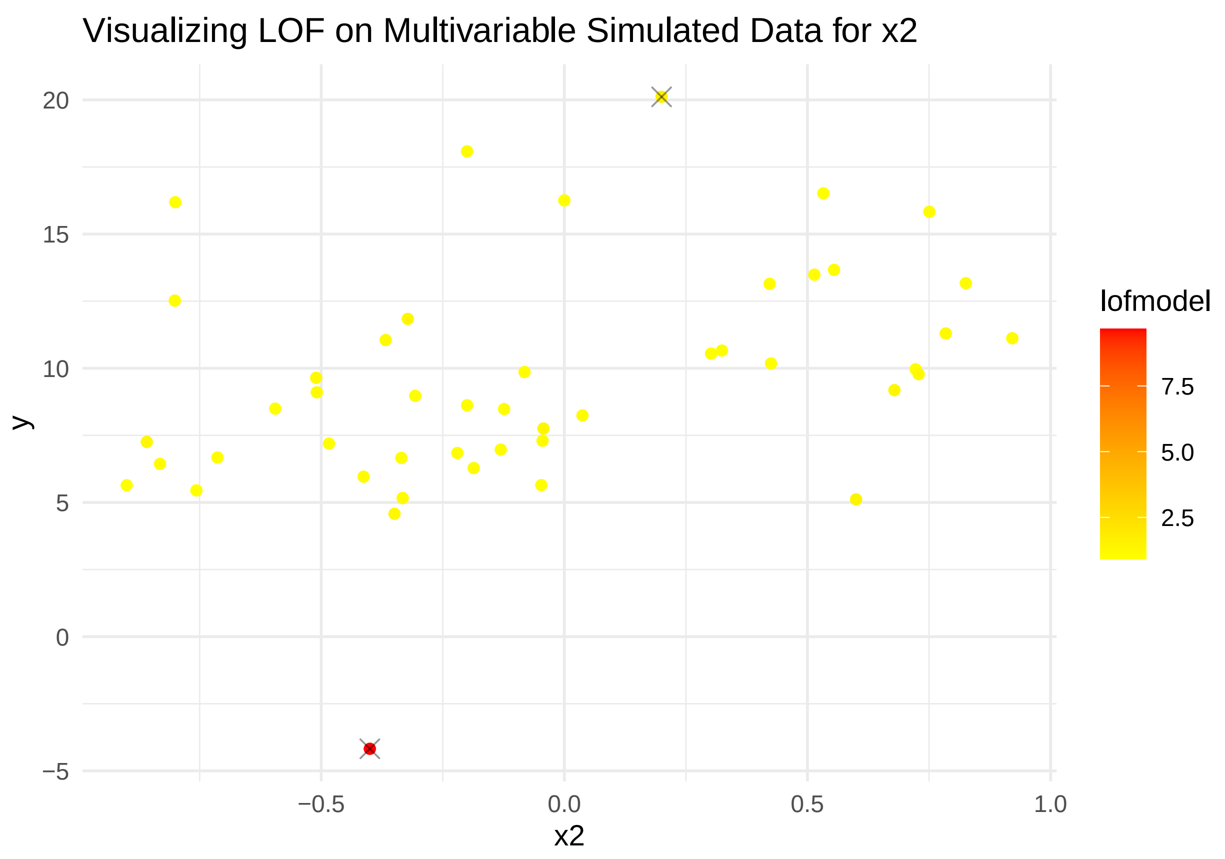
```r
# Calculate LOF for x1 and x2 on y
lofmodel = lof(complex_train, minPts = 5)
lof_outliers = which(lofmodel > 1.5)
lof_outliers
```

```
## [1] 46 47
```

```r
# Plot the data, x1 first
ggplot(complex_train, aes(x = x1complex, y = ycomplex)) +
  geom_point(aes(col=lofmodel)) +
  labs(title = "Visualizing LOF on Multivariable Simulated Data for x1", x = "x1", y = "y") +
  geom_point(data = complex_train[lof_outliers, ], shape = 4, size=3, alpha=0.4) +
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



```r
# Plot the data, x2 now
ggplot(complex_train, aes(x = x2complex, y = ycomplex)) +
  geom_point(aes(col=lofmodel)) +
  labs(title = "Visualizing LOF on Multivariable Simulated Data for x2", x = "x2", y = "y") +
  geom_point(data = complex_train[lof_outliers, ], shape = 4, size=3, alpha=0.4) +
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



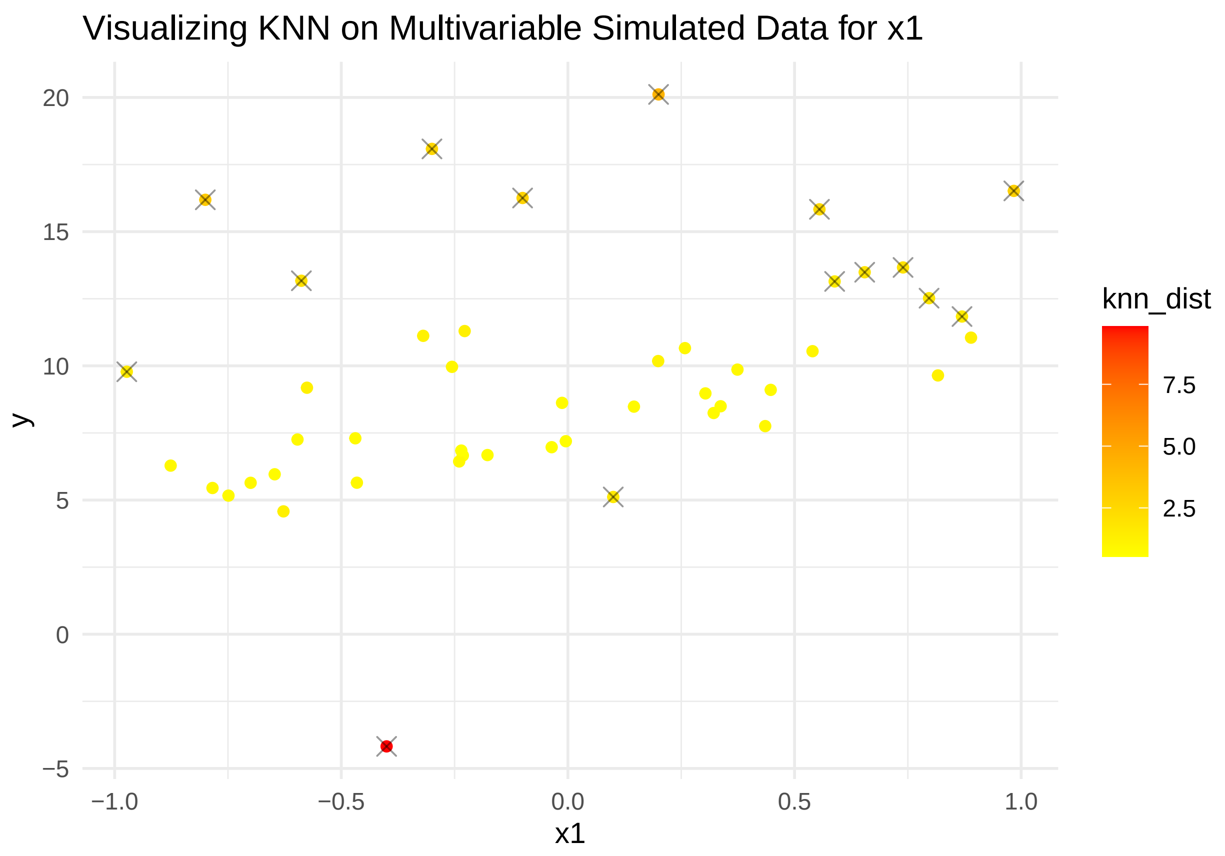
LOF was significantly less agressive here than in the other models, marking just the "obvious" ones.

Using KNN

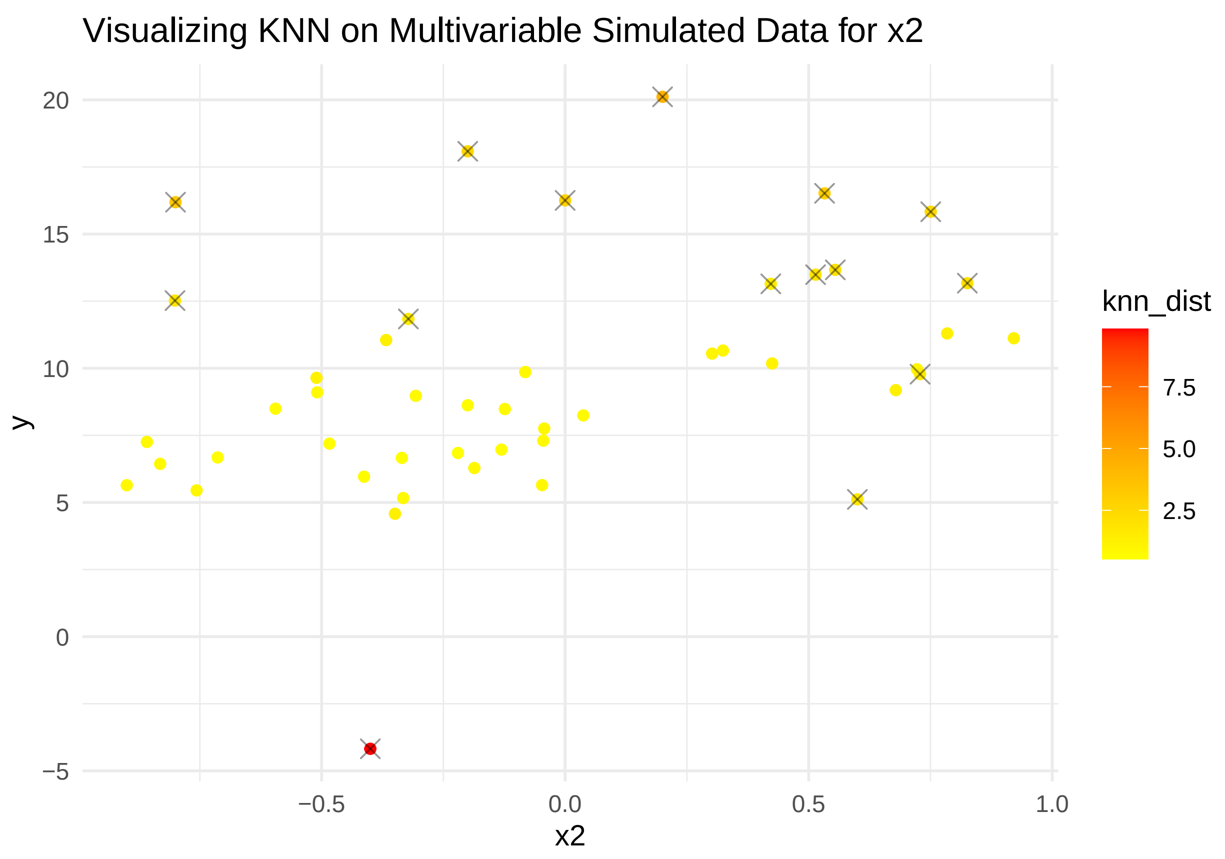
```
# Use KNN to detect outliers
knn_model = get.knn(complex_train, k = 5)
knn_dist = knn_model$nn.dist[,5]
knn_outliers = complex_train[knn_dist > 1.5, ]
knn_outliers
```

```
##      x1complex  x2complex  ycomplex
## 6   0.7967794 -0.8010677 12.520669
## 11 -0.5880509  0.8257518 13.170061
## 18  0.9838122  0.5326213 16.517999
## 20  0.5548904  0.7506427 15.831947
## 21  0.8694105 -0.3218541 11.837108
## 27 -0.9732193  0.7286789  9.779345
## 29  0.7393817  0.5546414 13.665035
## 35  0.6547466  0.5141743 13.486846
## 37  0.5884797  0.4222424 13.144460
## 42 -0.3000000 -0.2000000 18.083017
## 43 -0.1000000  0.0000000 16.253890
## 44  0.1000000  0.6000000  5.113406
## 45 -0.8000000 -0.8000000 16.186199
## 46 -0.4000000 -0.4000000 -4.179040
## 47  0.2000000  0.2000000 20.112512
```

```r
# Plot the data, x1 first
ggplot(complex_train, aes(x = x1complex, y = ycomplex)) +
  geom_point(aes(col=knn_dist)) +
  labs(title = "Visualizing KNN on Multivariable Simulated Data for x1", x = "x1", y = "y") +
  geom_point(data=knn_outliers, shape = 4, size=3, alpha=0.4) +
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



```r
# Plot the data, x2 now
ggplot(complex_train, aes(x = x2complex, y = ycomplex)) +
  geom_point(aes(col=knn_dist)) +
  labs(title = "Visualizing KNN on Multivariable Simulated Data for x2", x = "x2", y = "y") +
  geom_point(data=knn_outliers, shape = 4, size=3, alpha=0.4) +
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```

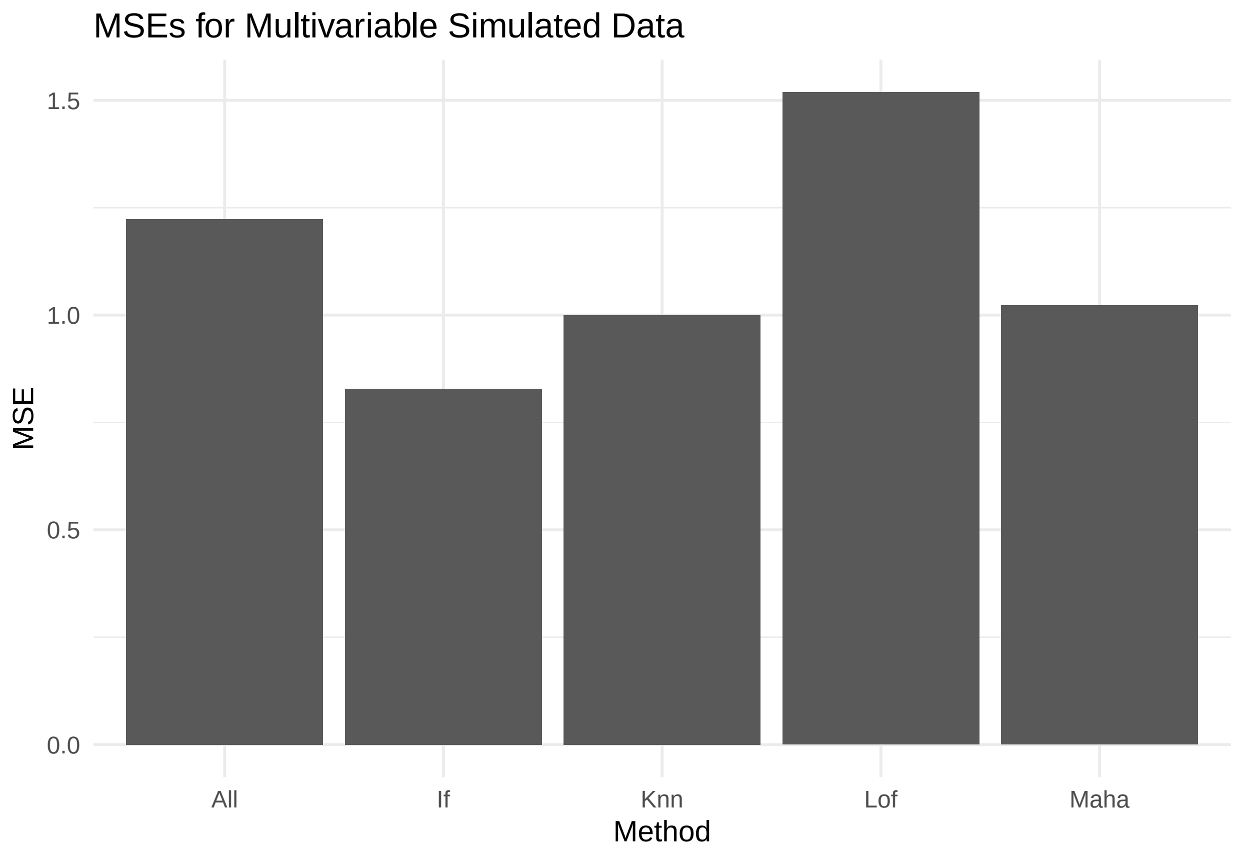


KNN is quite aggressive, marking points that are somewhat close to the distribution. One can also see the distance from each points' 5th nearest neighbor, so perhaps the points are looking at the outliers as their 5th nearest neighbor.

Time to generate multi-linear regression models for all these methods!

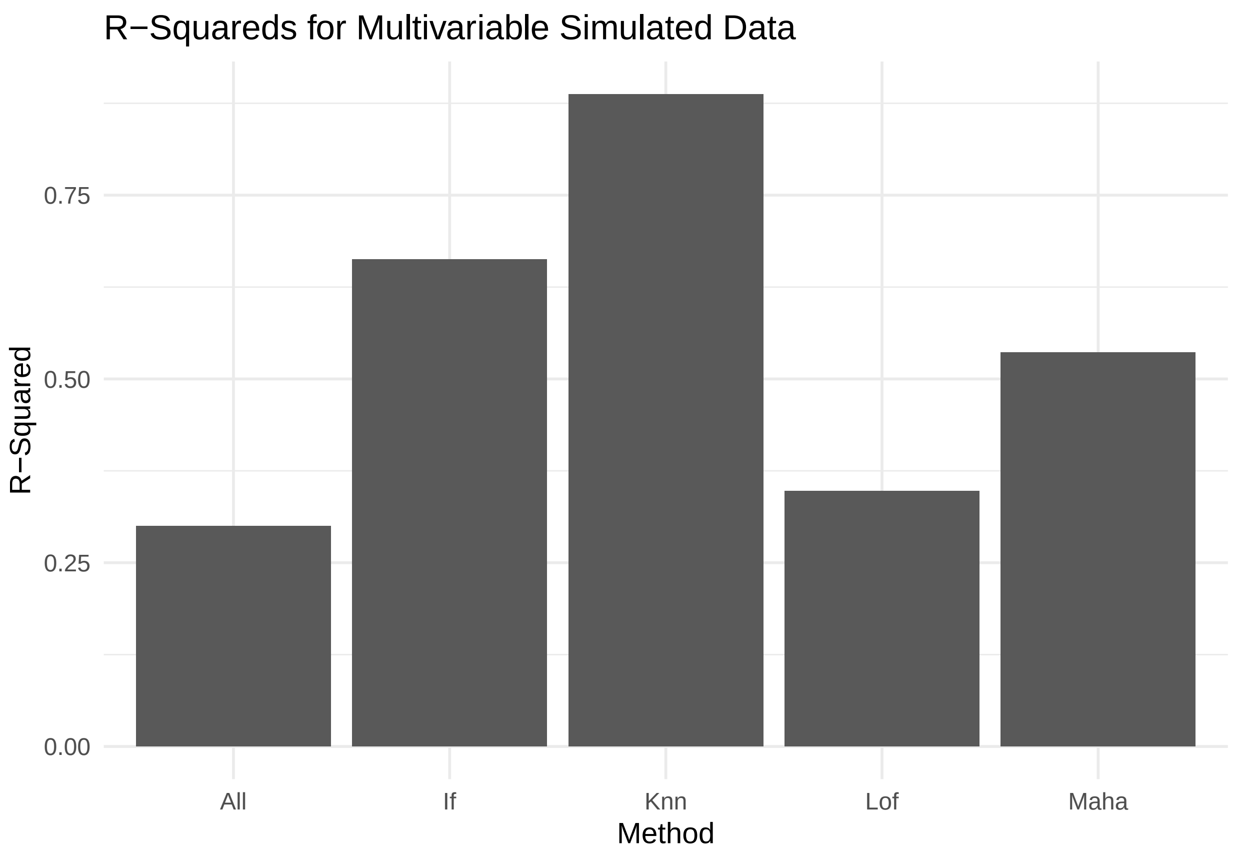
```
# Remove outliers from all methods
# Remove from mana based on index
complex_train_maha = complex_train[-maha_outliers, ]
# Remove from if based on index (if_outliers is a dataframe)
complex_train_if = complex_train[!complex_train$x1complex %in% if_outliers$x1complex, ]
# Remove from lof
complex_train_lof = complex_train[-lof_outliers, ]
# Remove from knn
complex_train_knn = complex_train[!complex_train$x1complex %in% knn_outliers$x1complex, ]

# Create a model with all data
model_all = lm(ycomplex~x1complex+I(x1complex^2)+x2complex+I(x2complex^2), data = complex_train)
# Create a model with mana outliers removed
model_maha = lm(ycomplex~x1complex+I(x1complex^2)+x2complex+I(x2complex^2), data = complex_train_maha)
# Create a model with if outliers removed
model_if = lm(ycomplex~x1complex+I(x1complex^2)+x2complex+I(x2complex^2), data = complex_train_if)
# Create a model with lof outliers removed
model_lof = lm(ycomplex~x1complex+I(x1complex^2)+x2complex+I(x2complex^2), data = complex_train_lof)
# Create a model with knn outliers removed
model_knn = lm(ycomplex~x1complex+I(x1complex^2)+x2complex+I(x2complex^2), data = complex_train_knn)

# Get the MSE for all models
```

```r
# All data
all_mse = mean((complex_test$ycomplex - predict(model_all, complex_test))^2)
# Mana
maha_mse = mean((complex_test$ycomplex - predict(model_maha, complex_test))^2)
# If
if_mse = mean((complex_test$ycomplex - predict(model_if, complex_test))^2)
# Lof
lof_mse = mean((complex_test$ycomplex - predict(model_lof, complex_test))^2)
# Knn
knn_mse = mean((complex_test$ycomplex - predict(model_knn, complex_test))^2)

# Plot MSEs
mse_df = data.frame(method = c("All", "Maha", "If", "Lof", "Knn"), mse = c(all_mse, maha_mse, if_mse, lo
ggplot(mse_df, aes(x = method, y = mse)) +
  geom_bar(stat = "identity") +
  labs(title = "MSEs for Multivariable Simulated Data", x = "Method", y = "MSE") +
  theme_minimal()
```



```r
# R-Squared now
# All data
all_r2 = summary(model_all)$r.squared
# Mana
maha_r2 = summary(model_maha)$r.squared
# If
if_r2 = summary(model_if)$r.squared
# Lof
```

```r
lof_r2 = summary(model_lof)$r.squared
# Knn
knn_r2 = summary(model_knn)$r.squared

# Plot R-Squareds
r2_df = data.frame(method = c("All", "Maha", "If", "Lof", "Knn"), r2 = c(all_r2, maha_r2, if_r2, lof_r2
ggplot(r2_df, aes(x = method, y = r2)) +
  geom_bar(stat = "identity") +
  labs(title = "R-Squareds for Multivariable Simulated Data", x = "Method", y = "R-Squared") +
  theme_minimal()
```



These findings are quite interesting. As before, LOF is quite bad for both MSE and R-Squared and Isolation Forest does quite well. However, it is also the case where KNN performs quite good, in fact better than isolation forest. Both IF and KNN seem to do much better than the baseline of Mahalanobis distance, which is something worth noting.

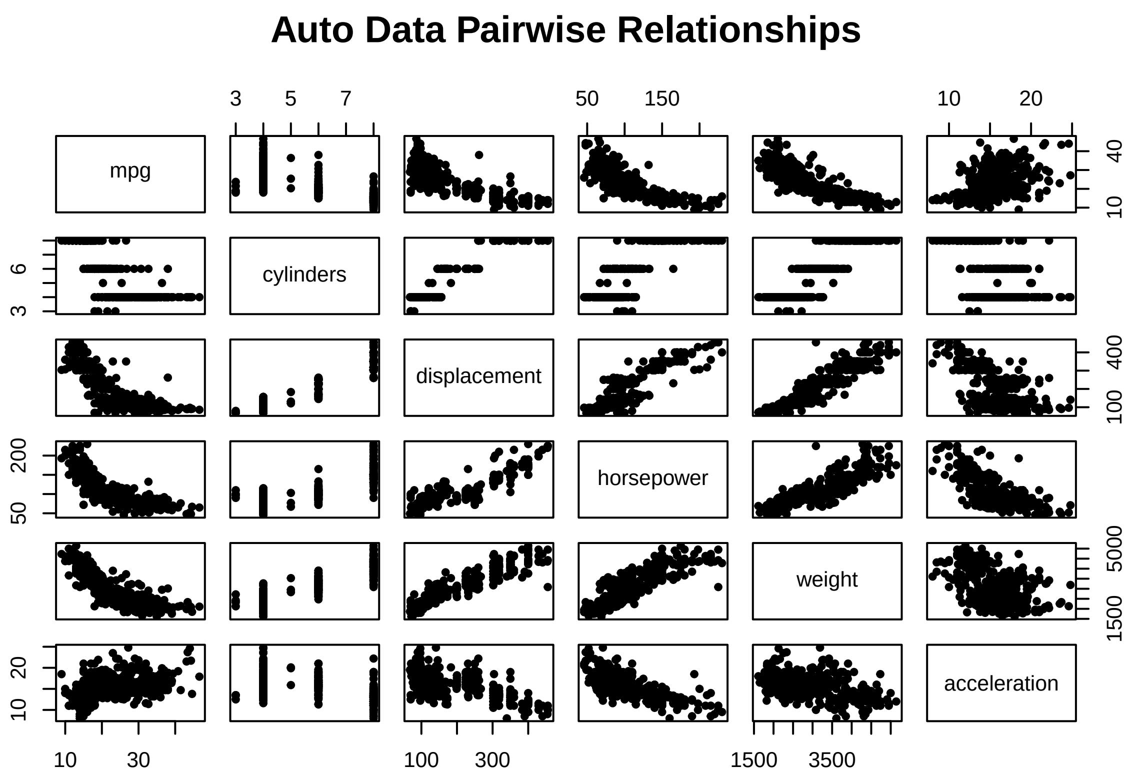
Real World Data:

```r
# Load the Auto data from ISLR package
auto_df = Auto[, 1:6] # Only use the numeric data

# Split the data into training and testing sets
train_idx = sample(1:nrow(auto_df), 0.8 * nrow(auto_df))
auto_train = auto_df[train_idx, ]
auto_test = auto_df[-train_idx, ]
```

In real world data, outliers are a common occurence. Whether from human error or uncommon occurence, outliers can be both a nuisance and an object of interest. We will use the Auto data set from the ISLR2 package, which contains data on 392 cars from the 1970s and 1980s. The data set contains 6 numeric variables: mpg, cylinders, displacement, horsepower, weight, and acceleration. We will use these variables to detect outliers in the data.

A potential use for outlier detection is finding outstanding cars: outliers in their performance and details. However, for the purposes of our testing, we will compare outlier detection methods by their ability to improve model training. By training a linear regression model on data without outliers, we will try to find if it is able to better fit the general trends of the data and make more accurate predictions. We split the Auto data set into a training and testing set to compare the performance of our models.

```r
# Plot the pairwise data relationships
plot(auto_df, pch = 20, main = "Auto Data Pairwise Relationships")
```



Many of the variables seem to be correlated, with some linear and some polynomial relationships. Looking at just the pairwise relationships, there are obviously at least a few outliers in the data. We can use Mahalanobis distance as a benchmark to detect outliers in the data. This will be used to compare our other methods' results.

Visualizations

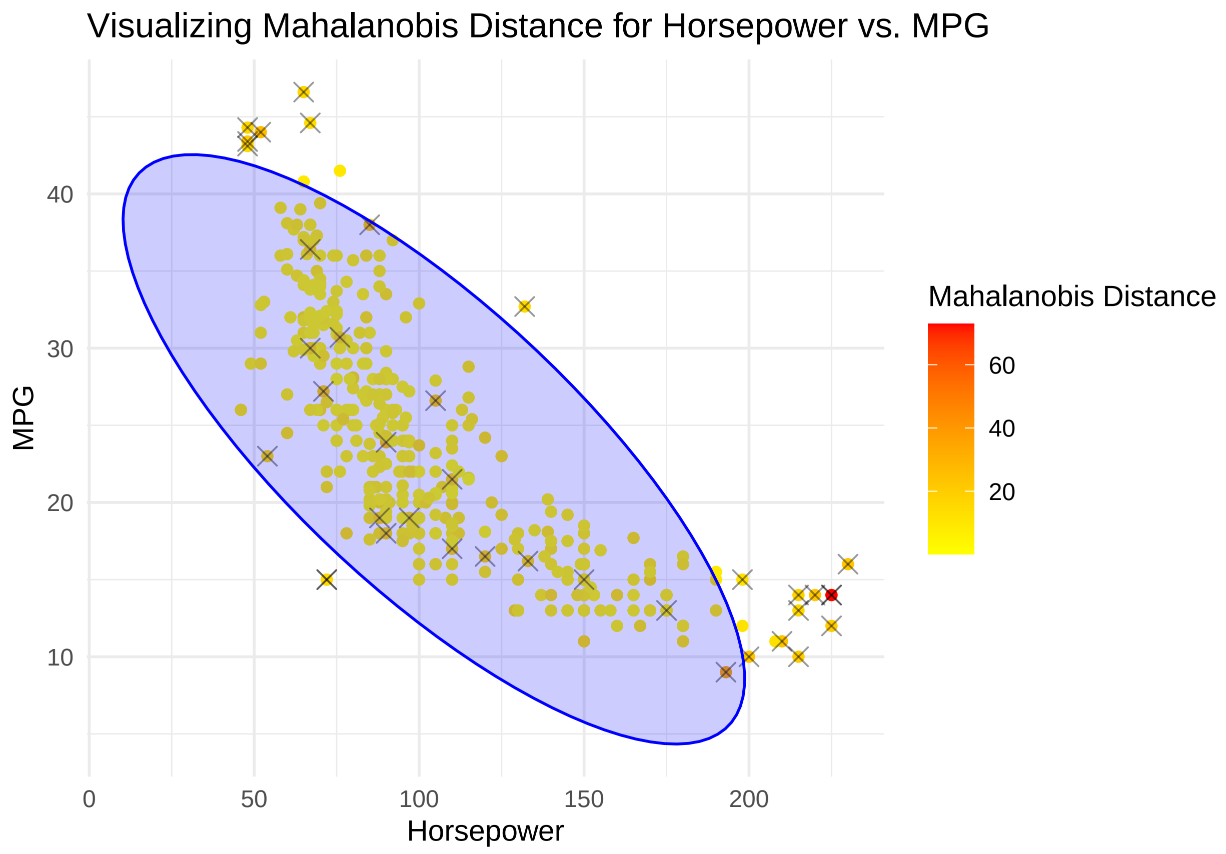
```r
# Test if the numeric data has any outliers according to Mahalanobis distance
auto_center = colMeans(auto_df)
auto_cov = cov(auto_df)
auto_dst = mahalanobis(auto_df, auto_center, auto_cov)
auto_cutoff = qchisq(0.95, df = ncol(auto_df))
```

```r
nrow(auto_df[auto_dst > auto_cutoff, ]) # Number of observations considered outliers
```

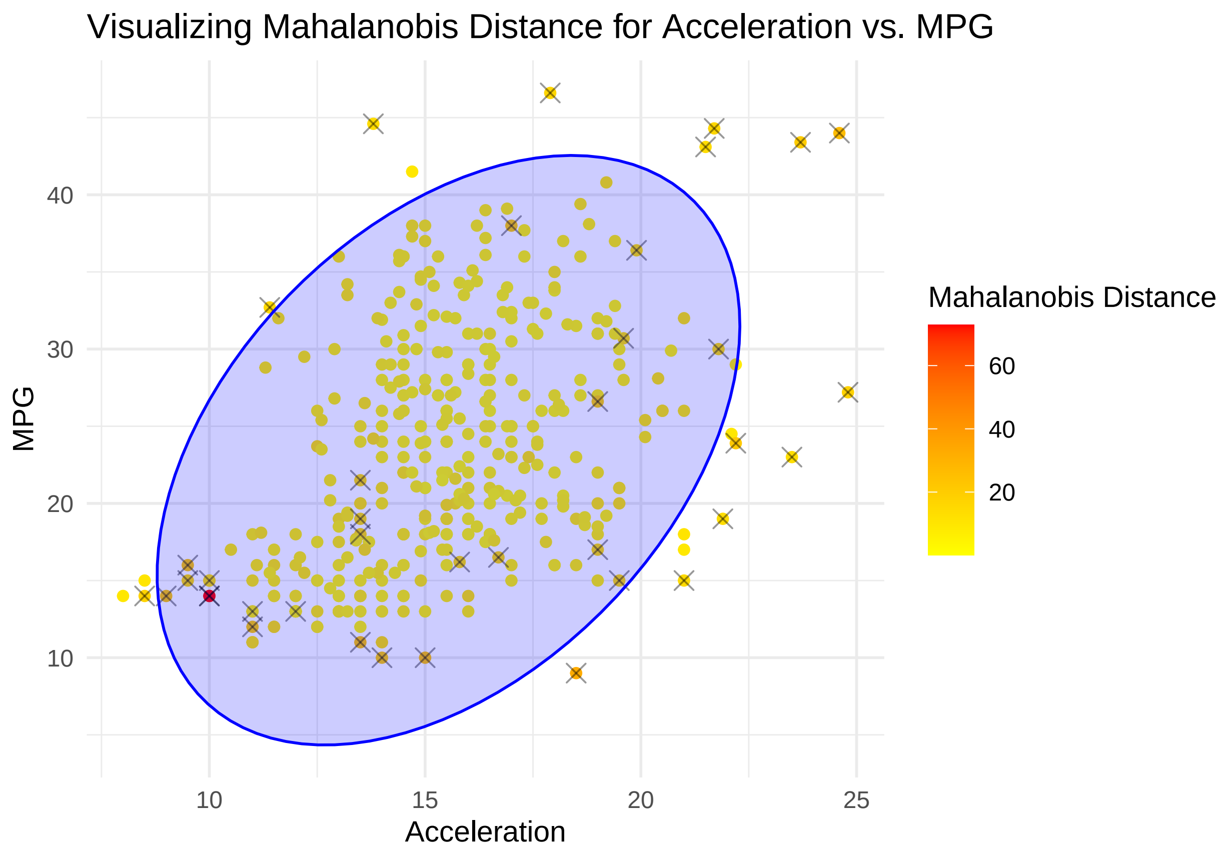
```
## [1] 38
```

Using Mahalanobis distance, we can see that there are 38 observations in the Auto data set that are considered outliers. We can visualize this by plotting the data and coloring the points based on their Mahalanobis distance. We can also plot the Mahalanobis distance ellipses for horsepower vs mpg and acceleration vs mpg, which gives us a reference for the outlier threshold if we were using just the two variables.

```r
# Create a scatter plot of horsepower vs mpg, coloring based on Mahalanobis distance
hp_ellipse = as.data.frame(ellipse(cor(auto_df[, c(4, 1)]), scale = c(sd(auto_df[,4]), sd(auto_df[,1]))

ggplot(auto_df, aes(x = horsepower, y = mpg)) +
  geom_point(aes(color=auto_dst)) +
  geom_point(data = auto_df[auto_dst > auto_cutoff, ], color = "black", shape = 4, size = 3, alpha=0.4)
  geom_polygon(data = hp_ellipse, alpha = 0.2, fill = "blue", color = "blue") +
  labs(title = "Visualizing Mahalanobis Distance for Horsepower vs. MPG", x = "Horsepower", y = "MPG",
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



```r
# Create a scatter plot of acceleration vs mpg, coloring based on Mahalanobis distance
acc_ellipse = as.data.frame(ellipse(cor(auto_df[, c(6, 1)]), scale = c(sd(auto_df[,6]), sd(auto_df[,1])

ggplot(auto_df, aes(x = acceleration, y = mpg)) +
  geom_point(aes(color=auto_dst)) +
  geom_point(data = auto_df[auto_dst > auto_cutoff, ], color = "black", shape = 4, size = 3, alpha=0.4)
  geom_polygon(data = acc_ellipse, alpha = 0.2, fill = "blue", color = "blue") +
  labs(title = "Visualizing Mahalanobis Distance for Acceleration vs. MPG", x = "Acceleration", y = "MPG
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal()
```



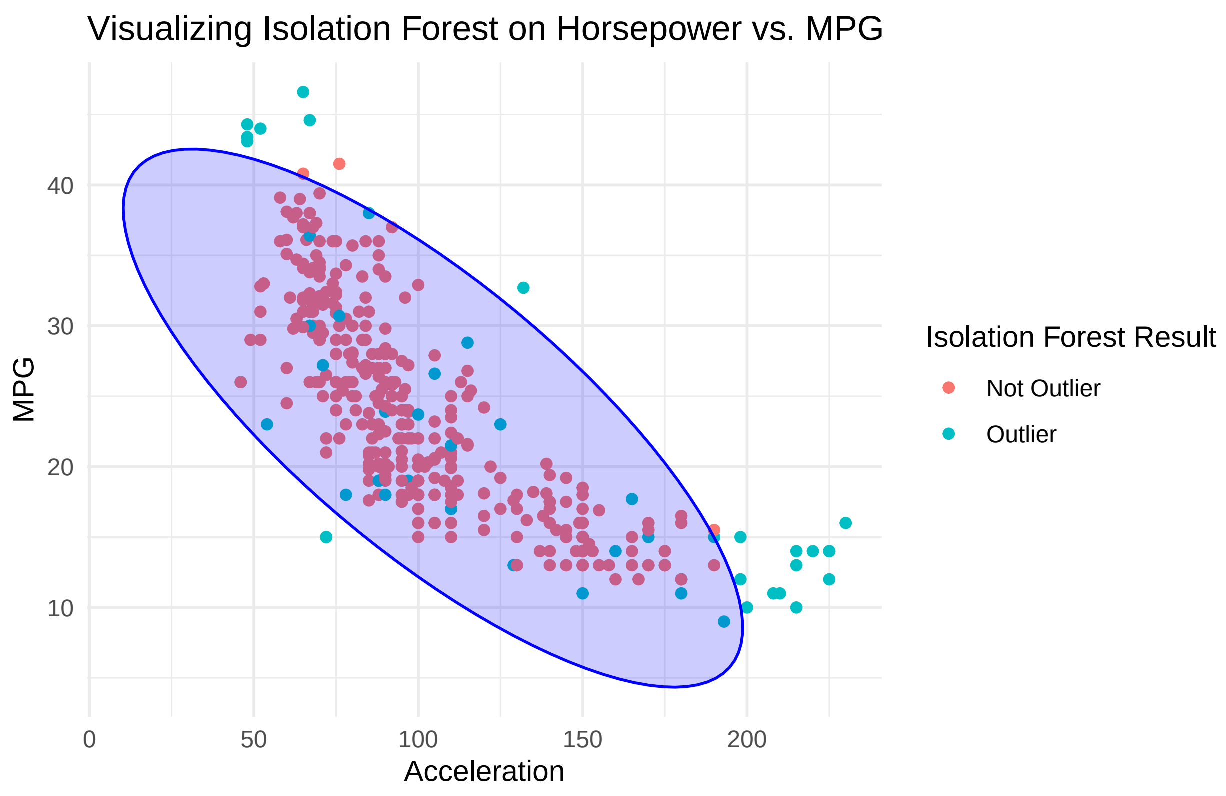
It's important to note that these visuals are only depicting the Mahalanobis distance thresholds for two variables. The other variables also contribute to a point's Mahalanobis distance, so some outliers may be inside the ellipse. Furthermore, the cutoff for considering a point an outlier is about 13, so orangish points are considered outliers. Overall, Mahalanobis distance seems to be robust and somewhat conservative. We can use this as a benchmark to compare our other methods' results.

```
# Perform isolation forest on the auto data
if_model = isolation.forest(auto_df, ntrees = 500, nthreads = 1)
if_pred = predict(if_model, auto_df)
if_outliers = as.factor(ifelse(if_pred > 0.5, "Outlier", "Not Outlier"))

length(if_outliers[if_outliers == "Outlier"]) # Number of outliers based on isolation forest
```

```
## [1] 52
```

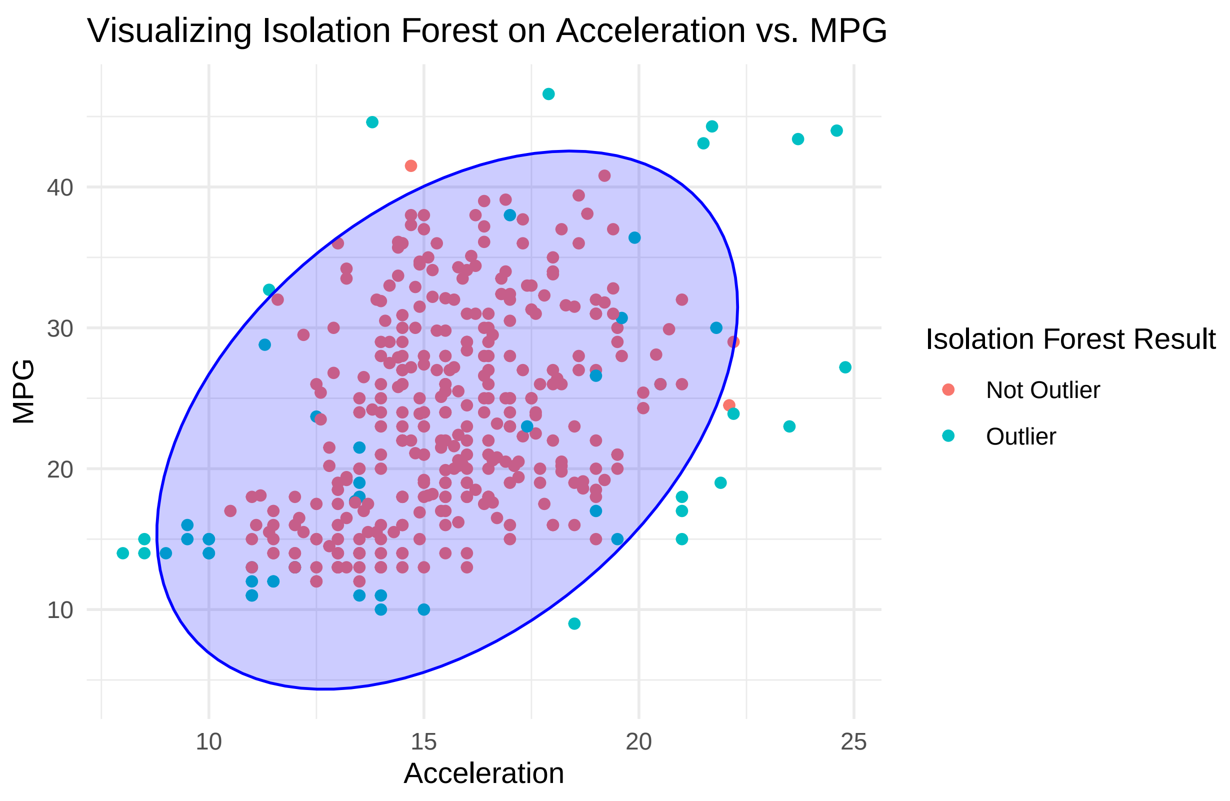
There are 49 outliers based on the isolation forest model. We can visualize this by plotting the data and coloring the points based on if the isolation forest predicted them as an outlier. We can also plot the Mahalanobis distance ellipses, which allows us to compare the isolation forest results to the Mahalanobis distance results.

```
# Plot the isolation forest outliers on horsepower vs mpg
ggplot(auto_df, aes(x = horsepower, y = mpg, color = if_outliers)) +
  geom_point() +
  geom_polygon(data = hp_ellipse, alpha = 0.2, color = "blue", fill = "blue") +
  labs(title = "Visualizing Isolation Forest on Horsepower vs. MPG", x = "Acceleration", y = "MPG", colo
  theme_minimal()
```



Blue ellipse represents Mahalanobis distance outlier threshold

```
# Plot the isolation forest outliers on acceleration vs mpg
ggplot(auto_df, aes(x = acceleration, y = mpg, color = if_outliers)) +
  geom_point() +
  geom_polygon(data = acc_ellipse, alpha = 0.2, fill = "blue", color = "blue") +
  labs(title = "Visualizing Isolation Forest on Acceleration vs. MPG", x = "Acceleration", y = "MPG", co
  theme_minimal()
```



Blue ellipse represents Mahalanobis distance outlier threshold

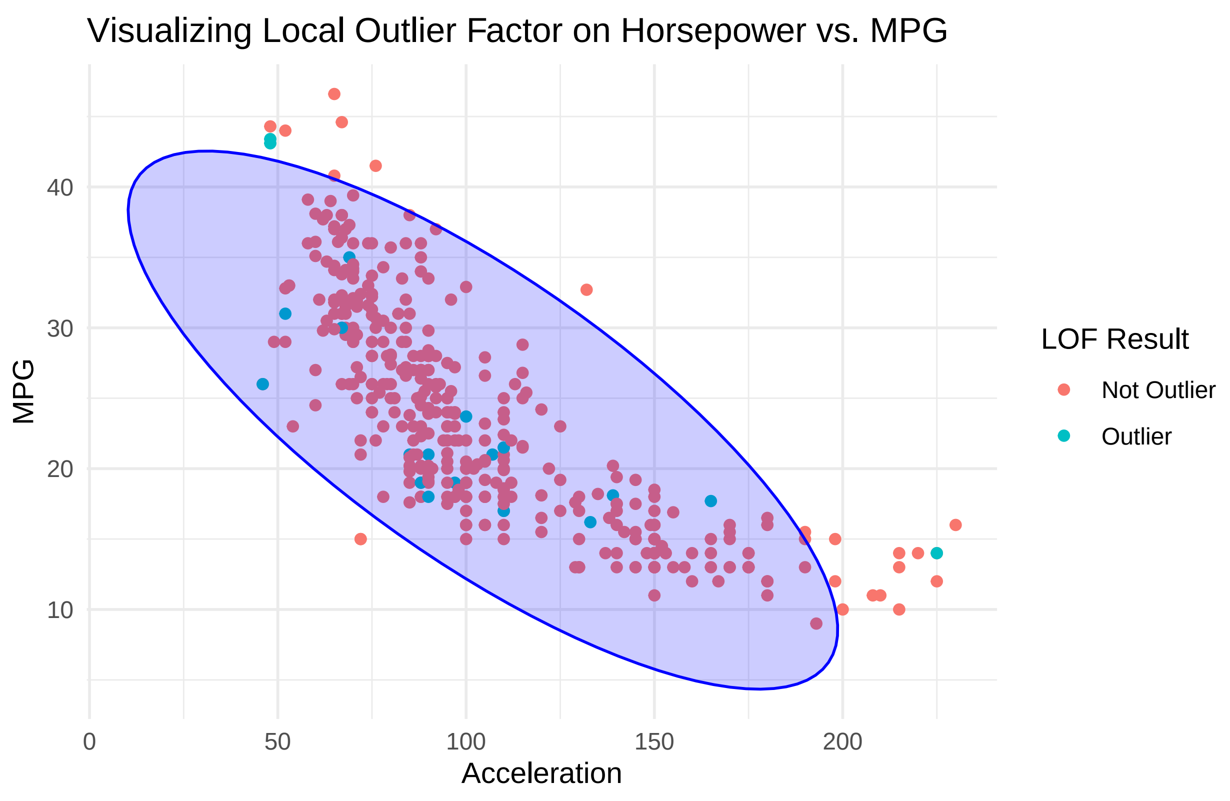
The results of using Isolation Forest is very similar to using Mahalanobis distance, with the exception of a few points. The Mahalanobis distance seems to be more conservative than the isolation forest results. This is likely due to the fact that isolation forest is based on the number of splits it takes to isolate a point, which is able to measure non-linear outliers between variables.

```
# Perform Local Outlier Factor on the auto data
lof_model = lof(auto_df, minPts = 6)
lof_outliers = as.factor(ifelse(lof_model > 1.5, "Outlier", "Not Outlier"))
length(lof_outliers[lof_outliers == "Outlier"]) # Number of observations considered outliers
```

```
## [1] 20
```

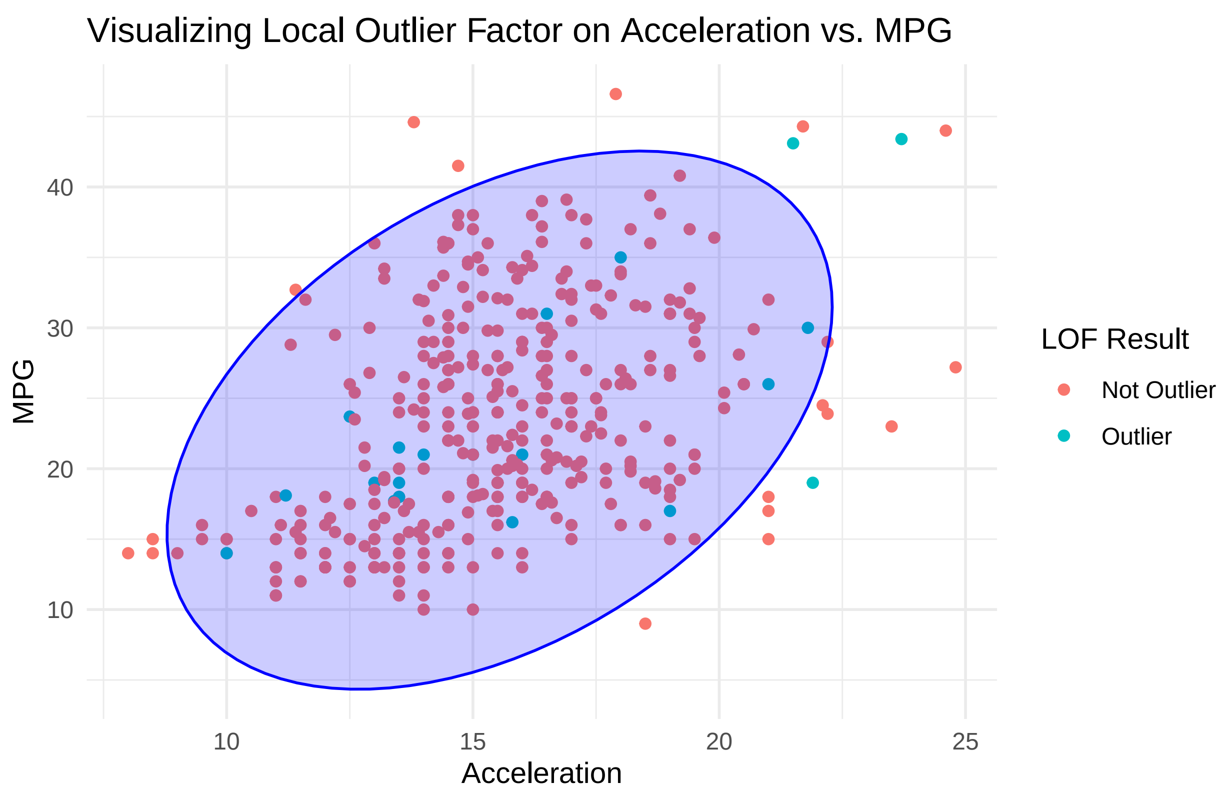
There are 20 outliers based on the LOF model. We can visualize this by plotting the data and coloring the points based on if the LOF predicted them as an outlier. We can also plot the Mahalanobis distance ellipses, which allows us to compare the LOF results to the Mahalanobis distance results.

```
# Plot the LOF outliers on horsepower vs mpg
ggplot(auto_df, aes(x = horsepower, y = mpg, color = lof_outliers)) +
  geom_point() +
  geom_polygon(data = hp_ellipse, alpha = 0.2, color = "blue", fill = "blue") +
  labs(title = "Visualizing Local Outlier Factor on Horsepower vs. MPG", x = "Acceleration", y = "MPG",
  theme_minimal()
```



Blue ellipse represents Mahalanobis distance outlier threshold

```r
# Plot the LOF outliers on acceleration vs mpg
ggplot(auto_df, aes(x = acceleration, y = mpg, color = lof_outliers)) +
  geom_point() +
  geom_polygon(data = acc_ellipse, alpha = 0.2, fill = "blue", color = "blue") +
  labs(title = "Visualizing Local Outlier Factor on Acceleration vs. MPG", x = "Acceleration", y = "MPG
  theme_minimal()
```



Blue ellipse represents Mahalanobis distance outlier threshold

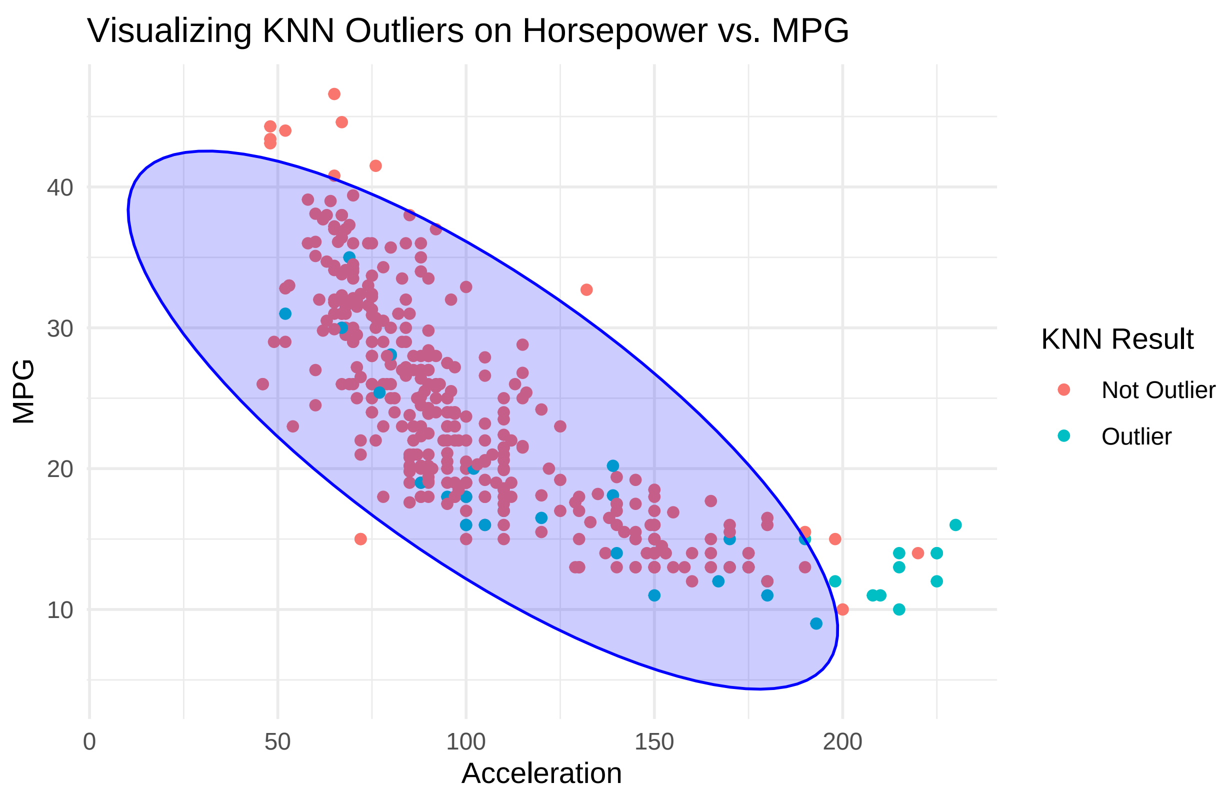
The results of LOF have subtle differences from the previous Isolation Forest and Mahalanobis distance methods. Firstly, LOF is much more conservative than the other two methods, which is especially relevant to the points outside of the ellipse. Due to its basis on local density, LOF does not consider groups of outliers as outliers. This can be seen in the bottom right of the horsepower vs mpg plot, where there is a dense group of points outside the ellipse. That group is aggressively considered outliers by Isolation Forest and Mahalanobis distance, but not LOF.

```
# Perform KNN on the auto data
knn_dist = get.knn(auto_df, k = 5)$nn.dist[, 5]
knn_outliers = which(knn_dist > 80)
knn_outliers = ifelse(knn_dist > 80, "Outlier", "Not Outlier")
length(knn_outliers[knn_outliers == "Outlier"])
```

```
## [1] 37
```

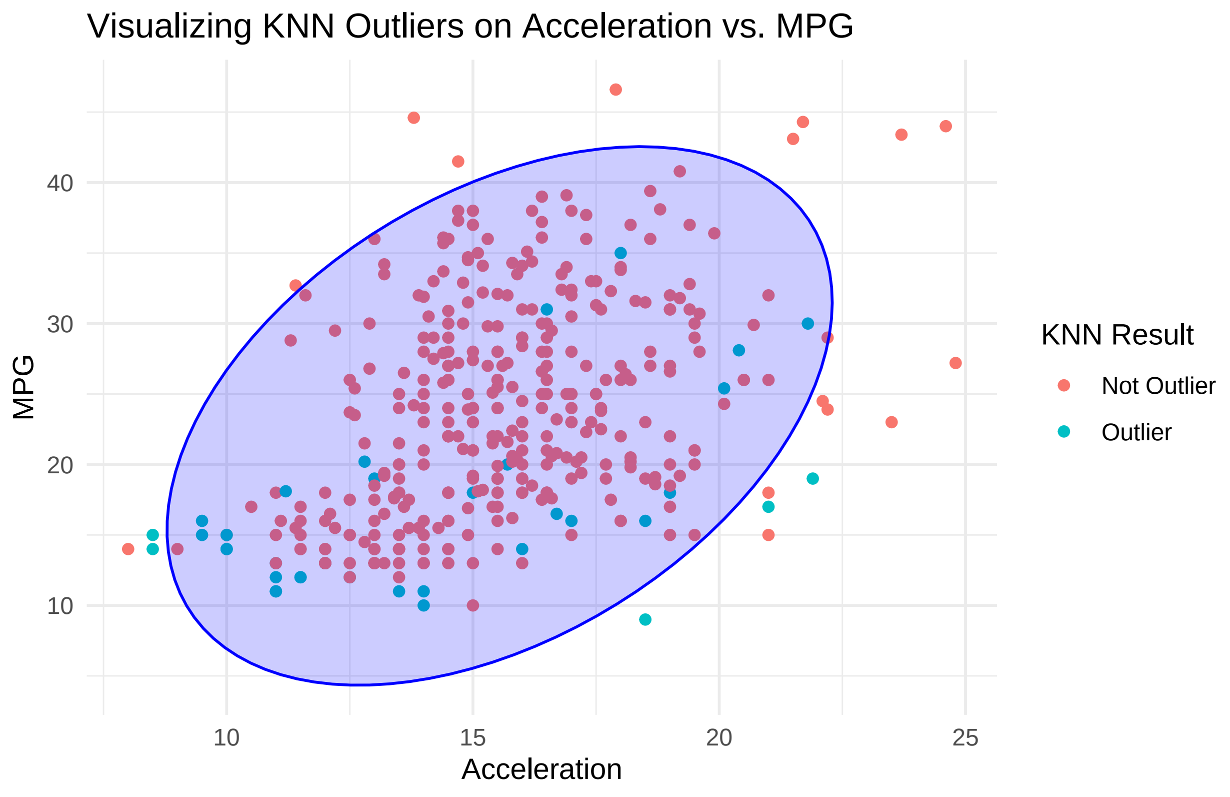
There are 37 outliers when using KNN with k = 5 with a threshold of 80. This means that any point with a distance greater than 80 to its 5th nearest neighbor is considered an outlier. We can visualize this by plotting the data and coloring the points based on if the KNN predicted them as an outlier. We can also plot the Mahalanobis distance ellipses, which allows us to compare the KNN results to the Mahalanobis distance results.

```
# Plot the KNN outliers on horsepower vs mpg
ggplot(auto_df, aes(x = horsepower, y = mpg, color = knn_outliers)) +
  geom_point() +
  geom_polygon(data = hp_ellipse, alpha = 0.2, color = "blue", fill = "blue") +
  labs(title = "Visualizing KNN Outliers on Horsepower vs. MPG", x = "Acceleration", y = "MPG", color =
  theme_minimal()
```



Blue ellipse represents Mahalanobis distance outlier threshold

```r
# Plot the KNN outliers on acceleration vs mpg
ggplot(auto_df, aes(x = acceleration, y = mpg, color = knn_outliers)) +
  geom_point() +
  geom_polygon(data = acc_ellipse, alpha = 0.2, fill = "blue", color = "blue") +
  labs(title = "Visualizing KNN Outliers on Acceleration vs. MPG", x = "Acceleration", y = "MPG", color
  theme_minimal()
```



Blue ellipse represents Mahalanobis distance outlier threshold

By tuning the threshold for KNN, it can be either very conservative or very aggressive. In this case, the threshold was set to be similar in aggressiveness as the other methods. This is an example of why hyper-parameter tuning is important, as varying the parameters for KNN can result in either too many or too few outliers detected. Additionally, the data was not scaled before performing KNN, which may have affected the results. Meanwhile, the other methods are not as affected by hyper-parameter tuning and scale of the data. Thus, KNN's results are not as robust, since some variables dominate its distance computations.

Model Performance Comparison

To measure the performance of the outlier detection methods, we will use each of them to remove outliers from a training data set. Then, we will use this data to train a linear regression model. To measure the effectiveness of the methods, we use each regression model to predict a testing set (with outliers) and compare their test MSE performance. We expect that pre-processing the data with the most effective outlier detection method will allow the model to learn the unskewed, general trends in the data, which will cause it to perform the best on the test set.

```r
# Create a linear regression model on the original train and test sets
original_lm = lm(mpg ~ ., data = auto_train)
original_pred = predict(original_lm, auto_test)
original_mse = mean((original_pred - auto_test$mpg)^2)
original_mse # Test MSE without outlier removal
```

```
## [1] 23.35558
```

Our benchmark for the performance of the outlier detection methods is the test MSE of a linear regression model trained on the original training set. We will compare the test MSE of the models trained on the outlier-removed training sets to this benchmark of 14.6 test MSE.

```r
# Compute the outliers in the training set using Mahalanobis distance
dst = mahalanobis(auto_train, colMeans(auto_train), cov(auto_train))
cutoff = quantile(auto_dst, 0.95)
mahal_outliers = which(dst > cutoff)

# Model with outliers removed using Mahalanobis distance
mahal_lm = lm(mpg ~ ., data = auto_train[dst <= cutoff, ])
mahal_pred = predict(mahal_lm, auto_test)
mahal_mse = mean((mahal_pred - auto_test$mpg)^2)
mahal_mse # Test MSE using Mahalanobis distance outlier removal
```

```
## [1] 25.71718
```

Mahalanobis distance outlier removal actually increases the test MSE of the model. The issue is that the outliers are not necessarily errors in the data set: they provide some information to the model. Outliers in the Auto data set are observations with the extreme ends of variable values, but they could potentially contribute to less common trends in the data. The reduction in MSE could also be because Mahalanobis distance is a linear measure of distance, and the outliers in the data set may be non-linear. This means that the Mahalanobis distance is not able to correctly capture all outliers in the training data set.

```r
# Compute the outliers in the training set using Isolation Forest
if_model = isolation.forest(auto_train, ntree = 500, nthreads = 1)
if_outliers = predict(if_model, auto_train)
if_outliers = as.factor(ifelse(if_outliers > .5, 1, 0))

# Model with outliers removed using isolation forest
if_lm = lm(mpg ~ ., data = auto_train[if_outliers == 0, ])
if_pred = predict(if_lm, auto_test)
if_mse = mean((if_pred - auto_test$mpg)^2)
if_mse # Test MSE using Mahalanobis distance outlier removal
```

```
## [1] 26.90539
```

Isolation Forest performs better than the original and Mahalanobis distance models. This supports the idea that some of the outliers in the training set are non-linear, making it difficult for Mahalanobis distance to identify them. That being said, Isolation Forest has the same problem such that some of the removed outliers may contribute to the underlying trends in the data, especially because it is much more aggressive in removing outliers. The results shown here are highly dependent on what outliers are left in the testing set.

```r
# Compute the outliers in the training set using LOF
lof_model = lof(auto_train, minPts = 6)

# Model with outliers removed using LOF
lof_lm = lm(mpg ~ ., data = auto_train[lof_model < 1.5,])
lof_pred = predict(lof_lm, auto_test)
lof_mse = mean((lof_pred - auto_test$mpg)^2)
lof_mse # Test MSE using Mahalanobis distance outlier removal
```

```
## [1] 24.1032
```

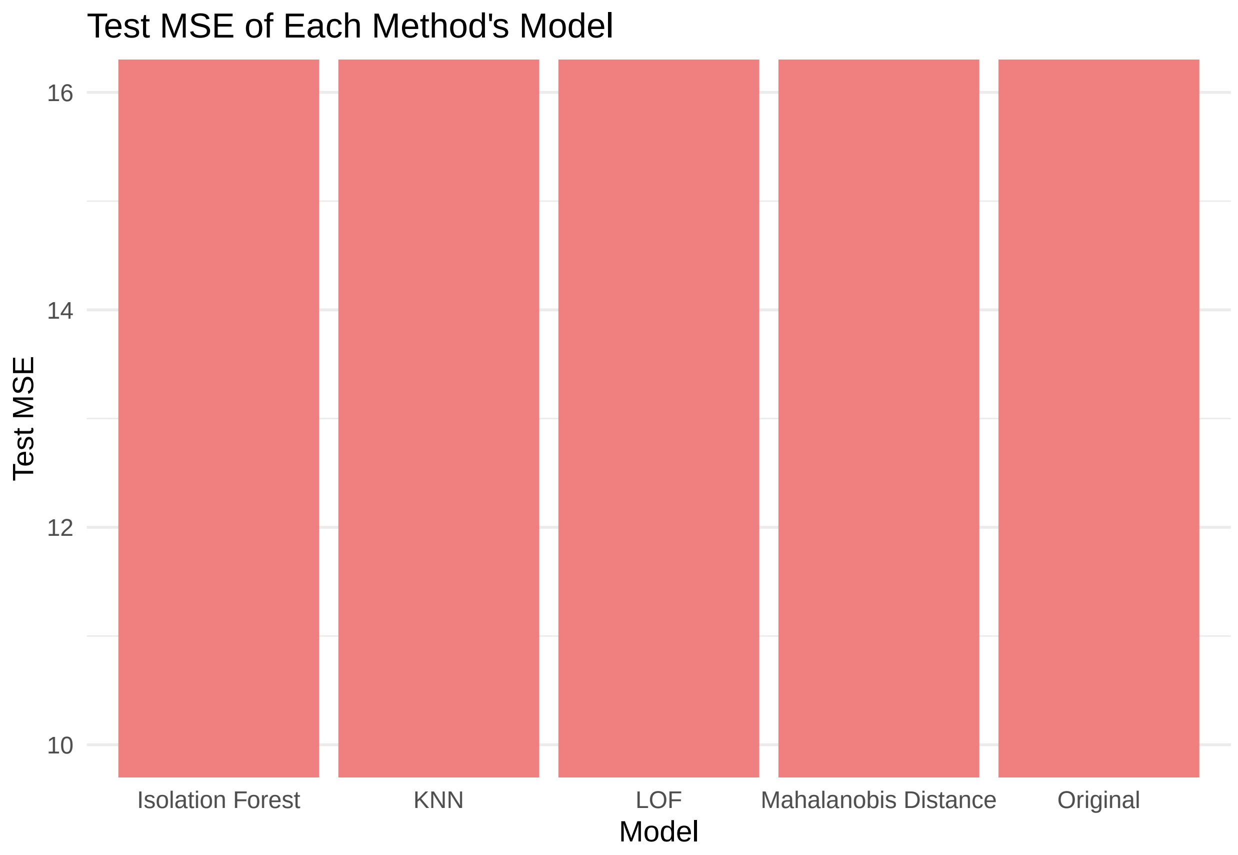
LOF performs better than the original and Mahalanobis distance models. However, it also performs slightly worse than the Isolation Forest. This makes sense because LOF is able to capture non-linear outliers in the data, similar to Isolation Forest. Additionally, LOF with the given parameters is less aggressive than Isolation Forest, which can sometimes prevent removing useful observations. Overall, LOF seems to perform similarly to Isolation Forest.

```r
# Compute the outliers in the training set using KNN
knn_dist = get.knn(auto_train, k = 5)$nn.dist[, 5]
knn_outliers = which(knn_dist > 80)

# Model with outliers removed using KNN
knn_lm = lm(mpg ~ ., data = auto_train[-knn_outliers, ])
knn_pred = predict(knn_lm, auto_test)
knn_mse = mean((knn_pred - auto_test$mpg)^2)
knn_mse # Test MSE using KNN outlier removal
```

```
## [1] 26.7072
```

KNN has the worst test MSE of all the methods. This is partially due to the hyper-parameter tuning, and other parameter combinations could produce more beneficial results. Furthermore, since the number of observations decreased from the train/testing split, KNN distances naturally increase for all points. Overall, since it performed worse than the original model, it means that the outliers it removed were useful observations for the model.

```r
# Plot the test MSE of each model
mse_df = data.frame(original_mse, mahal_mse, if_mse, lof_mse, knn_mse)
mse_df = gather(mse_df, key = "model", value = "mse")
ggplot(mse_df, aes(x = model, y = mse)) +
  geom_bar(stat = "identity", color = "lightcoral", fill = "lightcoral") +
  geom_hline(yintercept = original_mse, color = "black", linetype = "dashed") +
  labs(x = "Model", y = "Test MSE", title = "Test MSE of Each Method's Model") +
  theme_minimal() +
  scale_x_discrete(labels = c("Isolation Forest", "KNN", "LOF", "Mahalanobis Distance", "Original")) +
  coord_cartesian(ylim = c(10, 16))
```



From our results, we can see that Isolation Forest and LOF performed relatively the best. They had a slight improvement on the test MSE when we removed their outliers from the training set. This supports the idea that using outlier detection on training data can prevent overfitting and uncommon outliers from skewing the model. It is important to take these results with a grain of salt, however, because the results are highly dependent on the outliers that are left in the testing set. Furthermore, it is not always beneficial to remove extreme data points, because it can give important information to the model, and data removal is more suitable for observations known to be errors. Additionally, the results are dependent on the hyper-parameters chosen for each method. For example, KNN performed the worst, but it was also the least aggressive in removing outliers. If we had chosen a different set of hyper-parameters, it could have a vastly different performance.

We can further explore the results by comparing R squared. By using each method to remove outliers on the full data set, we can create a linear regression model and compare the R squared of each model. It is important to note that this is not a perfect comparison, because large amounts of removals would increase R squared by making the data as simple as possible. However, it can still give us a general idea of how well each method performs for finding general trends in the data.

```
# Compute the R squared of the model using the full data
original_lm = lm(mpg ~ ., data = auto_df)
original_r2 = summary(original_lm)$r.squared # R squared of original model
original_r2
```

```
## [1] 0.7076926
```

```
# Compute the R squared of the model using the full data without outliers from Mahalanobis distance
mahal_lm = lm(mpg ~ ., data = auto_df[auto_dst < auto_cutoff, ])
mahal_r2 = summary(mahal_lm)$r.squared # R squared of Mahalanobis distance model
mahal_r2
```

```
## [1] 0.7609626
```

```r
# Compute the R squared of the model using the full data without outliers from Isolation Forest
if_model = isolation.forest(auto_df, ntrees = 500, nthreads = 1)
if_pred = predict(if_model, auto_df)
if_lm = lm(mpg ~ ., data = auto_df[if_pred < .5, ])
if_r2 = summary(if_lm)$r.squared # R squared of Isolation Forest model
if_r2
```

```
## [1] 0.7587332
```

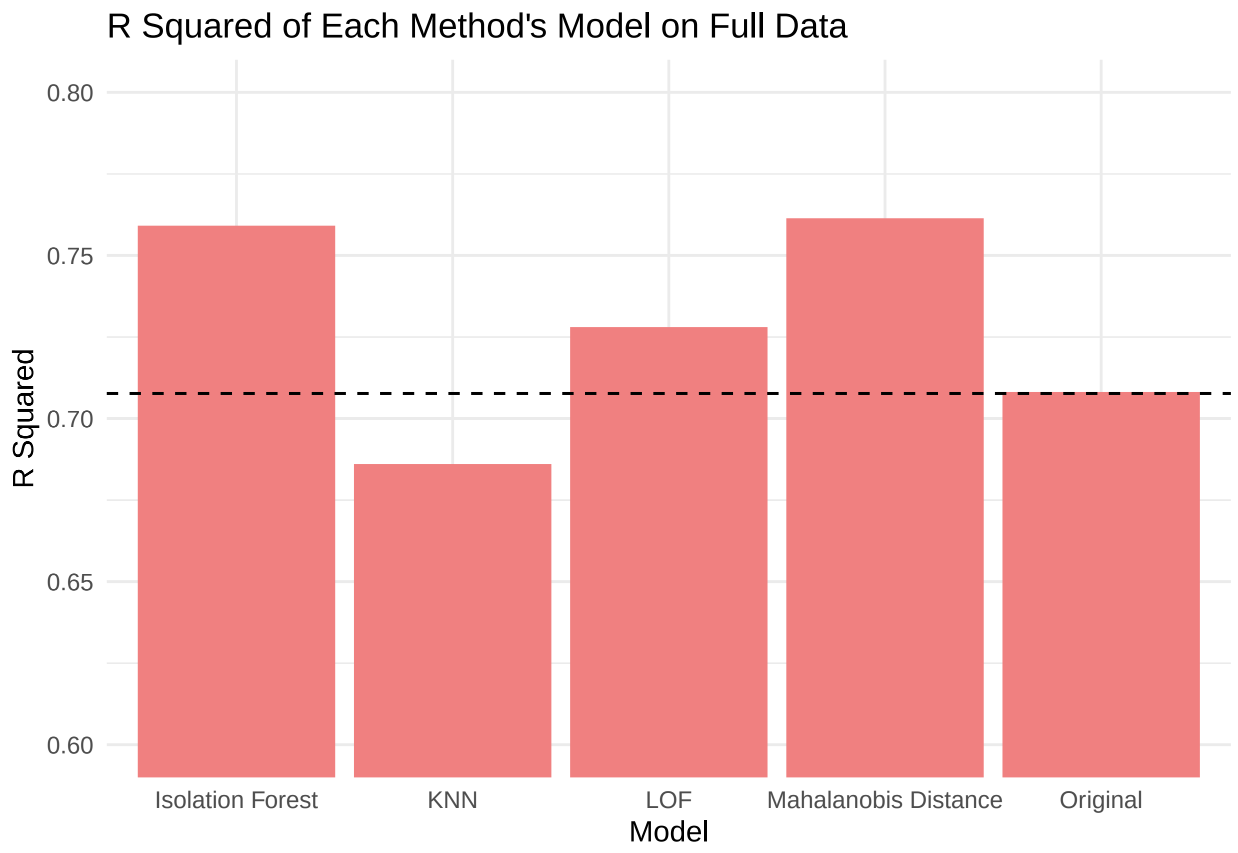
```r
# Compute the R squared of the model using the full data without outliers from Isolation Forest
lof_model = lof(auto_df, minPts = 6)
lof_lm = lm(mpg ~ ., data = auto_df[lof_model < 1.5, ])
lof_r2 = summary(lof_lm)$r.squared # R squared of Isolation Forest model
lof_r2
```

```
## [1] 0.7275611
```

```r
# Compute the R squared of the model using the full data without outliers from KNN
knn_result = get.knn(auto_df, k = 5)$nn.dist[, 5]
knn_outliers = which(knn_result > 80)
knn_lm = lm(mpg ~ ., data = auto_df[-knn_outliers, ])
knn_r2 = summary(knn_lm)$r.squared # R squared of KNN model
knn_r2
```

```
## [1] 0.6856
```

```r
# Plot the R squared of each model
r2_df = data.frame(original_r2, mahal_r2, if_r2, lof_r2, knn_r2)
r2_df = gather(r2_df, key = "model", value = "r2")
ggplot(r2_df, aes(x = model, y = r2)) +
  geom_bar(stat = "identity", color = "lightcoral", fill = "lightcoral") +
  geom_hline(yintercept = original_r2, color = "black", linetype = "dashed") +
  labs(x = "Model", y = "R Squared", title = "R Squared of Each Method's Model on Full Data") +
  theme_minimal() +
  scale_x_discrete(labels = c("Isolation Forest", "KNN", "LOF", "Mahalanobis Distance", "Original")) +
  coord_cartesian(ylim = c(0.6, 0.8))
```



From our results, we can see that Isolation Forest and Mahalanobis distance performed the best. They had a slight improvement on the R squared when we removed their outliers from the data. Again, it is natural that removing extreme points will improve the R squared of a model, but it does not mean the model is necessarily better. Being able to predict uncommon points is valuable depending on the data and scenario. Furthermore, since removing more points results in the data being easier to model, more aggressive methods will perform better. This is supported by the fact that Isolation Forest and Mahalanobis distance both have the best R squared and remove the most observations. Overall, each method has its pros and cons, and it is important to understand the data and scenario before choosing a method. Important factors to think about: the number of outliers, are the outliers errors or extreme points, how aggressive do you want to detect outliers, is the data linear or non-linear, etc. These answers will help choose the best outlier detection method.