Using an Ensemble Model to Predict Bike Share Demand in Seoul

Eli Markworth

December 2023

In this project, I will be examining the Seoul Bike Sharing Demand data set from the UCI Machine Learning Repository. I will be making an ensemble model using kNN, linear regression, and decision tree regression to predict the number of bike-share bikes rented in any given hour in Seoul, South Korea. The data set contains information about the number of bikes rented per hour for each day from December 2017 to the end of November 2018. There is also information on weather and holiday status for each entry.

Data Aquisition

The libraries I am using are being loaded in a hidden code chunk.

To load the data in, I am using `read_csv` from the tidyverse, as I will be using many more tidyverse functions later on. This also allows me to set the language encoding to be compatible, as the data was collected from a country that does not use the Latin alphabet. I used `str` and `head` to make sure it loaded in all the variables and cells correctly.

Data Exploration

```
str(bikedata)
```

```
## spc_tbl_ [8,760 × 14] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Date                     : chr [1:8760] "01/12/2017" "01/12/2017" "01/12/2017"
"01/12/2017" ...
##  $ Rented Bike Count        : num [1:8760] 254 204 173 107 78 100 181 460 930 490
...
##  $ Hour                     : num [1:8760] 0 1 2 3 4 5 6 7 8 9 ...
##  $ Temperature(°C)          : num [1:8760] -5.2 -5.5 -6 -6.2 -6 -6.4 -6.6 -7.4
-7.6 -6.5 ...
##  $ Humidity(%)              : num [1:8760] 37 38 39 40 36 37 35 38 37 27 ...
##  $ Wind speed (m/s)         : num [1:8760] 2.2 0.8 1 0.9 2.3 1.5 1.3 0.9 1.1 0.5
...
##  $ Visibility (10m)         : num [1:8760] 2000 2000 2000 2000 2000 ...
##  $ Dew point temperature(°C): num [1:8760] -17.6 -17.6 -17.7 -17.6 -18.6 -18.7 -1
9.5 -19.3 -19.8 -22.4 ...
##  $ Solar Radiation (MJ/m2)  : num [1:8760] 0 0 0 0 0 0 0 0 0.01 0.23 ...
##  $ Rainfall(mm)             : num [1:8760] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Snowfall (cm)            : num [1:8760] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Seasons                  : chr [1:8760] "Winter" "Winter" "Winter" "Winter" ...
##  $ Holiday                  : chr [1:8760] "No Holiday" "No Holiday" "No Holiday"
"No Holiday" ...
##  $ Functioning Day          : chr [1:8760] "Yes" "Yes" "Yes" "Yes" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Date = col_character(),
##   ..   `Rented Bike Count` = col_double(),
##   ..   Hour = col_double(),
##   ..   `Temperature(°C)` = col_double(),
##   ..   `Humidity(%)` = col_double(),
##   ..   `Wind speed (m/s)` = col_double(),
##   ..   `Visibility (10m)` = col_double(),
##   ..   `Dew point temperature(°C)` = col_double(),
##   ..   `Solar Radiation (MJ/m2)` = col_double(),
##   ..   `Rainfall(mm)` = col_double(),
##   ..   `Snowfall (cm)` = col_double(),
##   ..   Seasons = col_character(),
##   ..   Holiday = col_character(),
##   ..   `Functioning Day` = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
bikedata.tf <- bikedata

for (i in c(2,4,5,6,7,8,9,10,11)) {
  bikedata.tf[,i] <- scale(bikedata[,i])
}
```

Doing a quick overview of the data with `str`, we see that the `Date` variable in a character type. I will need to change this to a proper date/time format. The variables `Seasons`, `Holiday`, and `Functioning Day` are all as characters, but would be better as factors. I am normalizing my continuous features now so that my examination of the data will be more meaningful. The normalized data is going in a tibble `bikedata.tf`.

```
bikedata$Date <- dmy(bikedata$Date)
bikedata.tf$Date <- dmy(bikedata.tf$Date)
summary(bikedata.tf$Date)
```

```
##         Min.      1st Qu.       Median         Mean      3rd Qu.         Max.
## "2017-12-01" "2018-03-02" "2018-06-01" "2018-06-01" "2018-08-31" "2018-11-30"
```

```
bikedata %>%
  group_by(Hour) %>%
  summarize(sum(n = n()))
```

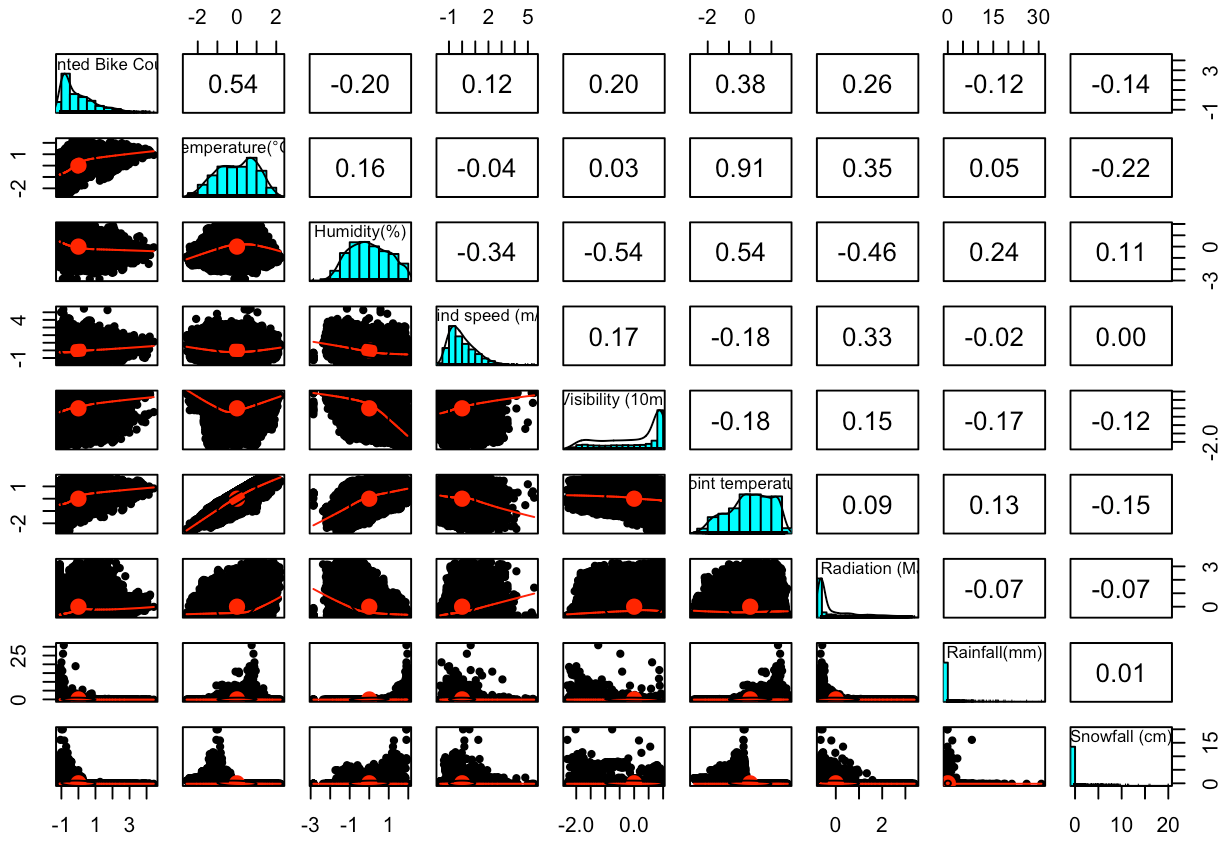
```
## # A tibble: 24 × 2
##     Hour `sum(n = n())`
##    <dbl>          <int>
##  1     0            365
##  2     1            365
##  3     2            365
##  4     3            365
##  5     4            365
##  6     5            365
##  7     6            365
##  8     7            365
##  9     8            365
## 10     9            365
## # ℹ 14 more rows
```

I changed the date entries to a proper date format: year, month, day. We can see that the first day in the data set in the first of December 2017, and the last day is November 30th, 2018. This means we have no erroneous dates added to the data set and there are complete sets of hours for each day.

The distribution of Rented Bike Counts, Visibility, Solar radiation, Rainfall, and Snowfall all look like they may not be normally distributed when plotted in histograms. The others look normally distributed enough that they will be suitable for our linear regression model. The kNN and Decision Tree models will be robust to non-normally distributed data, and the linear regression needs a close enough distribution. There is no exact cutoff of normal to non-normal distribution which makes a linear regression model stop working. It will work better or worse, but either way it will make predictions.

We can evaluate the non-normality of the data using a Shapiro-Wilk test. The variables Rented Bike Count, Wind Speed, Visibility, Solar Radiation, Rainfall, and Snowfall are significantly non-normal. I did not include the data that looked normally distributed enough in the histogram chunk, as those are good enough for linear regression. I had to test only a sample of the data because the Shapiro-Wilk test can only handle a vector up to 5000, and our data is greater.

```
pairs.panels(bikedata.tf[,c(2,4,5,6,7,8,9,10,11)])
```



We can check the co-linearity of the variables with the `pairs.panels` function. We see that many of the variables have weak relationships with each other. Some that do stick out are Rented Bike Count with Temperature, Temperature with Dewpoint Temp., and Humidity with Temp. and Visibility. There is high correlation in those variables, which could cause problems.

```
# I'm keeping Rented Bike Count observations that are outliers, as it is the dependen
t variable
for (i in 4:11) {
  bikedata.tf <- bikedata.tf[which(bikedata.tf[,i] < 3 & bikedata.tf[,i] > -3), ]
}
```

I am removing observations with outliers which are defined as more than 3 standard deviations from the mean. I am using this definition because others, such as using 1.5 times the inter quartile range, would remove too many values from our features which are not normally distributed.

```
bikedata.tf$Seasons <- as.factor(bikedata.tf$Seasons)
table(bikedata.tf$Seasons)
```

```
##
## Autumn Spring Summer Winter
##   2118   2126   2117   1991
```

```
bikedata.tf$Holiday <- as.factor(bikedata.tf$Holiday)
table(bikedata.tf$Holiday)
```

```
##
##    Holiday No Holiday
##         425       7927
```

```
bikedata.tf$`Functioning Day` <- as.factor(bikedata.tf$`Functioning Day`)
table(bikedata.tf$`Functioning Day`)
```

```
##
##   No  Yes
##  288 8064
```

By turning all the character features into factors, I can double check that they are entered in correctly by using the `table` function. It does not look like there are any typos.

Data Cleaning and Shaping

```
for (i in 1:ncol(bikedata.tf)) {
  nas <- length(which(is.na(bikedata.tf[,i]) == TRUE))
  print(paste("Feature", i, "has", nas, "NA values."))
}
```

```
## [1] "Feature 1 has 0 NA values."
## [1] "Feature 2 has 0 NA values."
## [1] "Feature 3 has 0 NA values."
## [1] "Feature 4 has 0 NA values."
## [1] "Feature 5 has 0 NA values."
## [1] "Feature 6 has 0 NA values."
## [1] "Feature 7 has 0 NA values."
## [1] "Feature 8 has 0 NA values."
## [1] "Feature 9 has 0 NA values."
## [1] "Feature 10 has 0 NA values."
## [1] "Feature 11 has 0 NA values."
## [1] "Feature 12 has 0 NA values."
## [1] "Feature 13 has 0 NA values."
## [1] "Feature 14 has 0 NA values."
```

```
for (i in c(2, 4:11)) {
  na_inds <- sample(1:nrow(bikedata.tf), size = floor(0.05 * nrow(bikedata.tf)))
  bikedata.tf[na_inds,i] <- NA
}
```

The source of this data set notes that there are no missing values in the data, but I like double checking. None of the features have missing/NA values. Because there are no missing values, I will remove some at

random to show how to deal with them. I convert 5% of values in continuous variables to NAs. I am not converting values in categorical variables because those variables are things like Date, Hour, and Season, which are integral to the way the data is collected. They would therefore only be missing in extreme cases. If there are missing values, their replacement would be easy to determine given the context of the data but that replacement would be repetitive/simple enough that it would be inappropriate to include in this project (I would likely manually impute those data).

```
bikedata.tf$`Rented Bike Count`[is.na(bikedata.tf$`Rented Bike Count`)] <- median(bik
edata.tf$`Rented Bike Count`, na.rm = T)

bikedata.tf$`Temperature(°C)`[is.na(bikedata.tf$`Temperature(°C)`)] <- median(bikedat
a.tf$`Temperature(°C)`, na.rm = T)

bikedata.tf$`Humidity(%)`[is.na(bikedata.tf$`Humidity(%)`)] <- median(bikedata.tf$`Hu
midity(%)`, na.rm = T)

bikedata.tf$`Wind speed (m/s)`[is.na(bikedata.tf$`Wind speed (m/s)`)] <- median(biked
ata.tf$`Wind speed (m/s)`, na.rm = T)

bikedata.tf$`Visibility (10m)`[is.na(bikedata.tf$`Visibility (10m)`)] <- median(biked
ata.tf$`Visibility (10m)`, na.rm = T)

bikedata.tf$`Dew point temperature(°C)`[is.na(bikedata.tf$`Dew point temperature
(°C)`)] <- median(bikedata.tf$`Dew point temperature(°C)`, na.rm = T)

bikedata.tf$`Solar Radiation (MJ/m2)`[is.na(bikedata.tf$`Solar Radiation (MJ/m2)`)]
<- median(bikedata.tf$`Solar Radiation (MJ/m2)`, na.rm = T)

bikedata.tf$`Rainfall(mm)`[is.na(bikedata.tf$`Rainfall(mm)`)] <- median(bikedata.tf$`
Rainfall(mm)`, na.rm = T)

bikedata.tf$`Snowfall (cm)`[is.na(bikedata.tf$`Snowfall (cm)`)] <- median(bikedata.tf
$`Snowfall (cm)`, na.rm = T)
```

I am imputing the data by median, as mean imputation would further skew data in features not normally distributed.

```
encSeasons <- model.matrix(~bikedata.tf$Seasons - 1)
encHoliday <- model.matrix(~bikedata.tf$Holiday - 1)
encFuncDay <- model.matrix(~bikedata.tf$`Functioning Day` - 1)

bikedata.en <- bikedata.tf[,1:11]

bikedata.en <- cbind(bikedata.en, encSeasons, encHoliday, encFuncDay)
```

Our kNN model needs its categorical variables dummy encoded. I am making a new data frame that holds the dummy values so that I don't have to do any column selection with the models that don't need the encoded values. I am using the `model.matrix` function to create simple dummy encoding.

To transform our first variable `Rented Bike Count` (RBC) into a distribution that is more normal, I am going

to try square-root, log, and inverse transformation methods. The RBC data looks slightly skewed right. When I assess its normality using the Shapiro-Wilk test, it is non-normal at sampling sizes of 5000 and 500. With a square root transformation, we can detect a slight improvement in the normality, as the p-value increases very slightly. From this, it seems that if I wish to detect if any of my transformations improve the data, I'll need to use a smaller sample size (500). I am using an offset on the data for the square root and log transformations, as those functions are incompatible with certain values such as negatives and 0. Using a log transformation, it seems two of the transformations are better the square-root transformation in terms of propensity to increase the normality of the data - the smaller offsets. The inverse transformation seems inappropriate for this data. The best transform in this case is the log with small offset.

Similar to how I approached transforming the RBC variable, I assessed each transformation for the Wind Speed variable by Shapiro-Wilk p-value. The log transformation worked best, and I did not choose any other offset, as the data should behave the same way due to its right skew.

None of the transformations I explored for the Visibility variable were sufficient at normalizing the Visibility data. I will have to continue with imperfect data.

Likewise with the prior transformation, no transformations were able to normalize the Solar Radiation data.

This is the same for the Rainfall variable.

And the Snowfall variable.

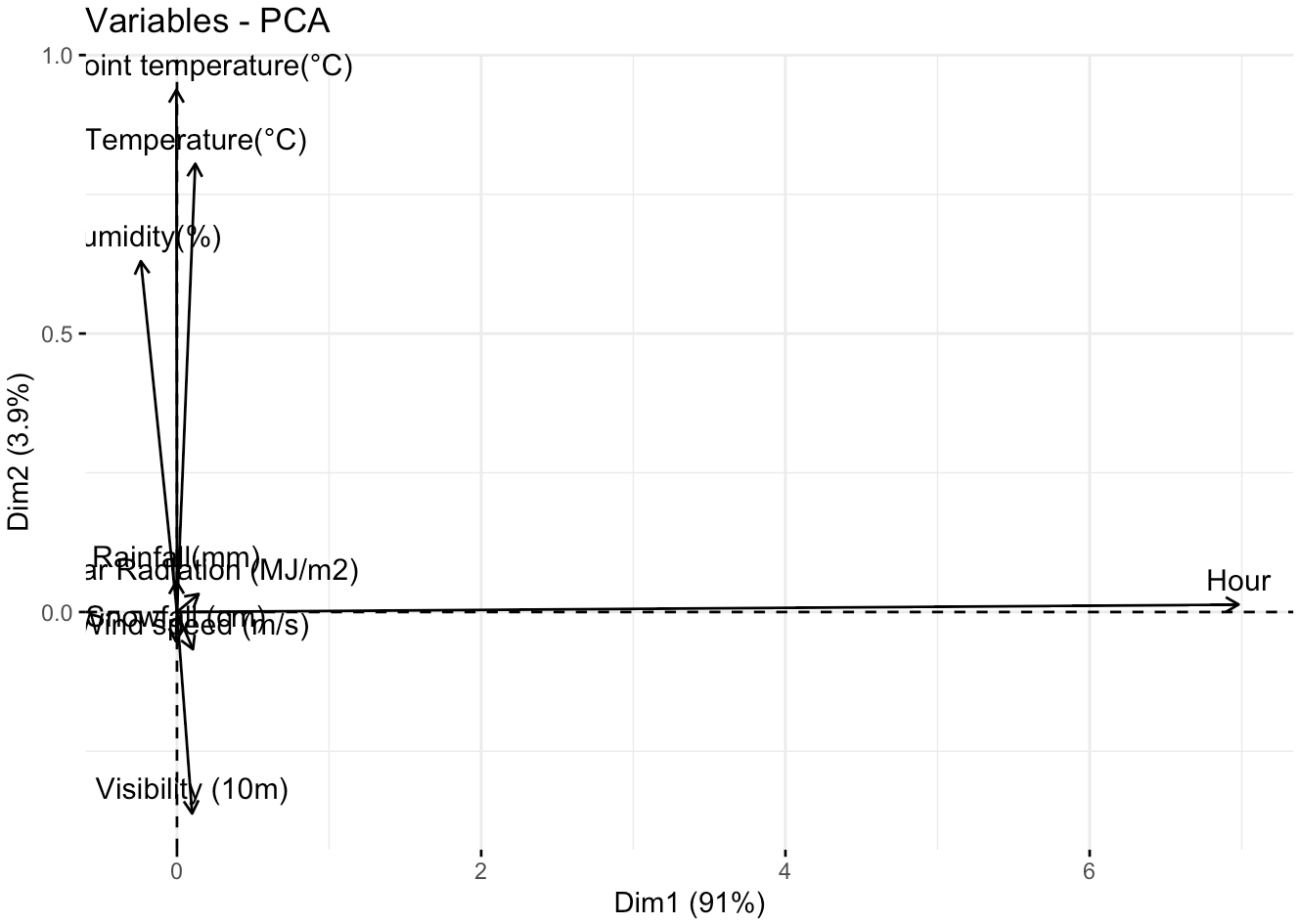
```
bike.pca <- princomp(bikedata.tf[,3:11])
summary(bike.pca)
```

```
## Importance of components:
##                           Comp.1     Comp.2     Comp.3     Comp.4     Comp.5
## Standard deviation     6.9873341 1.43753726 1.20499982 0.86928126 0.50601863
## Proportion of Variance 0.9101716 0.03852463 0.02706913 0.01408708 0.00477346
## Cumulative Proportion  0.9101716 0.94869622 0.97576535 0.98985243 0.99462589
##                            Comp.6      Comp.7    Comp.8       Comp.9
## Standard deviation     0.314785572 0.269144348 0.2601112 0.2215586939
## Proportion of Variance 0.001847268 0.001350426 0.0012613 0.0009151196
## Cumulative Proportion  0.996473155 0.997823581 0.9990849 1.0000000000
```

```
bike.pca$loadings[,1]
```

```
##                      Hour           Temperature(°C)              Humidity(%)
##              0.9988486111              0.0172946248             -0.0339913818
##          Wind speed (m/s)          Visibility (10m) Dew point temperature(°C)
##              0.0150360068              0.0142781165             -0.0005234181
##   Solar Radiation (MJ/m2)              Rainfall(mm)             Snowfall (cm)
##              0.0203744676             -0.0006023121             -0.0011101028
```

```
fviz_pca_var(bike.pca)
```



We can get peek at what variables are going to have to most effect on what features may explain to most variance in the data. I'm excluding the Date and categorical variables, as they are incompatible with this principal component analysis. I am using the `princomp` function to run the PCA as it is in a package that is already downloaded. I'm using a handy function `fviz_pca_var` to help us visualize the PCA. From the summary, we see that an overwhelming majority of the variance is explained by the first principal component. We also see that the `Hour` variable has a much higher loading value than other features in the data. The visualization of the PCA helps us connect these dots by showing just how dominant the hour variable is in the data. Temperature and Dew Point Temp. are runner ups.

```
bikedata$Precipitation <- ifelse(bikedata$`Rainfall(mm)` > 0 | bikedata$`Snowfall (c
m)` > 0, 1, 0)

bikedata.tf$Precipitation <- ifelse(bikedata.tf$`Rainfall(mm)` >
                                      min(bikedata.tf$`Rainfall(mm)`) |
                                      bikedata.tf$`Snowfall (cm)` >
                                      min(bikedata.tf$`Snowfall (cm)`), 1, 0)

bikedata.en$Precipitation <- ifelse(bikedata.en$`Rainfall(mm)` >
                                      min(bikedata.en$`Rainfall(mm)`) |
                                      bikedata.en$`Snowfall (cm)` >
                                      min(bikedata.en$`Snowfall (cm)`), 1, 0)
```

I am making a new feature called "precipitation", which is a binary factor variable describing whether there is precipitation or not. Any hour with snowfall or rainfall above 0 will count as a entry with precipitation.

Model Construction

```
dl <- nrow(bikedata.tf)
train_inds <- sample(1:dl, size = floor(.8*dl))

ben.train <- bikedata.en[train_inds,]
ben.test <- bikedata.en[-train_inds,]

btf.train <- bikedata.tf[train_inds,]
btf.test <- bikedata.tf[-train_inds,]
```

I am splitting my data with 80:20 training to testing. I chose this split because it is fairly standard and I will be using different splits later on in Model Tuning & Performance Improvement anyway.

```
# Make Date variable the number of days since 1970
ben.train$Date <- as.numeric(ben.train$Date) / 86400
ben.test$Date <- as.numeric(ben.test$Date) / 86400

initial_k <- sqrt(nrow(ben.train))
knnmodel <- knnreg(ben.train[,-2], ben.train[,2], k = initial_k)
knnpredictions.en.tf <- predict(knnmodel, ben.test[,-2])
```

I am using `knnreg` from the caret package to make my k Nearest Neighbors (kNN) model as it is handy for doing kNN for regression where most handle classification. I am starting with a k of 81.7373843, as it is the square root of the number of observations in the training data set. I am using the training data set with encoded values for the categorical features.

```
dtmodel <- rpart(`Rented Bike Count` ~ ., data = btf.train)

# Make predictions
dtpredictions.tf <- predict(dtmodel, btf.test[,-2])
```

I am using the `rpart` function in the rpart package to create my decision tree model. An alternative package I could use is `RWeka` but I find it less intuitive in the way it handles the data it is given.

```
mlrmodel <- lm(`Rented Bike Count` ~., data = btf.train)
summary(mlrmodel)
```

```
##
## Call:
## lm(formula = `Rented Bike Count` ~ ., data = btf.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.89890 -0.17794 -0.01285  0.15671  0.90964
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  8.6428024  2.2364296   3.865 0.000112 ***
## Date                        -0.0005021  0.0001255  -4.000 6.41e-05 ***
## Hour                         0.0190335  0.0005133  37.079  < 2e-16 ***
## `Temperature(°C)`            0.1132932  0.0101761  11.133  < 2e-16 ***
## `Humidity(%)`               -0.0974042  0.0066098 -14.736  < 2e-16 ***
## `Wind speed (m/s)`           0.0315288  0.0105624   2.985 0.002846 **
## `Visibility (10m)`           0.0001194  0.0042075   0.028 0.977371
## `Dew point temperature(°C)`  0.0666200  0.0109306   6.095 1.16e-09 ***
## `Solar Radiation (MJ/m2)`   -0.0069747  0.0044940  -1.552 0.120705
## `Rainfall(mm)`              -0.1186565  0.0172163  -6.892 6.00e-12 ***
## `Snowfall (cm)`              0.1521784  0.0168395   9.037  < 2e-16 ***
## SeasonsSpring               -0.1956964  0.0251614  -7.778 8.51e-15 ***
## SeasonsSummer               -0.1224124  0.0167679  -7.300 3.20e-13 ***
## SeasonsWinter               -0.4571266  0.0367104 -12.452  < 2e-16 ***
## HolidayNo Holiday            0.0967089  0.0150973   6.406 1.60e-10 ***
## `Functioning Day`Yes         0.7851850  0.0192299  40.831  < 2e-16 ***
## Precipitation               -0.2598019  0.0218693 -11.880  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2675 on 6664 degrees of freedom
## Multiple R-squared:  0.6105, Adjusted R-squared:  0.6096
## F-statistic: 652.8 on 16 and 6664 DF,  p-value: < 2.2e-16
```

```
mlrmodel <- lm(`Rented Bike Count` ~. - `Solar Radiation (MJ/m2)` - `Visibility (10
m)`,
               data = btf.train)

# Model Preview
summary(mlrmodel)
```

```
##
## Call:
## lm(formula = `Rented Bike Count` ~ . - `Solar Radiation (MJ/m2)` -
##     `Visibility (10m)`, data = btf.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.89604 -0.17740 -0.01464  0.15703  0.90932
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  8.4408387  2.2314582   3.783 0.000157 ***
## Date                        -0.0004906  0.0001253  -3.917 9.07e-05 ***
## Hour                         0.0191150  0.0005102  37.467  < 2e-16 ***
## `Temperature(°C)`            0.1096795  0.0099150  11.062  < 2e-16 ***
## `Humidity(%)`               -0.0942850  0.0057997 -16.257  < 2e-16 ***
## `Wind speed (m/s)`           0.0278466  0.0102623   2.713 0.006675 **
## `Dew point temperature(°C)`  0.0665956  0.0109218   6.097 1.14e-09 ***
## `Rainfall(mm)`              -0.1190040  0.0171943  -6.921 4.90e-12 ***
## `Snowfall (cm)`              0.1515849  0.0168300   9.007  < 2e-16 ***
## SeasonsSpring               -0.1943567  0.0249568  -7.788 7.86e-15 ***
## SeasonsSummer               -0.1199729  0.0166934  -7.187 7.35e-13 ***
## SeasonsWinter               -0.4549473  0.0364926 -12.467  < 2e-16 ***
## HolidayNo Holiday            0.0962941  0.0150917   6.381 1.88e-10 ***
## `Functioning Day`Yes         0.7847702  0.0192282  40.814  < 2e-16 ***
## Precipitation               -0.2605727  0.0218623 -11.919  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2675 on 6666 degrees of freedom
## Multiple R-squared:  0.6104, Adjusted R-squared:  0.6095
## F-statistic: 745.8 on 14 and 6666 DF,  p-value: < 2.2e-16
```

```
mlrpredictions.tf <- predict(mlrmodel, btf.test[,-2])
```

I am using the base R function `lm` to create my Multiple Linear Regression (MLR) model. The first iteration finds a model to predict RBCs with all other variables in the data. However, some of those variables have low p-values, and may just be noise. I can fix this by re-creating the model, this time without the variables that aren't likely to contribute to accurate predictions.

Model Evaluation

If we are to evaluate the models, we have a few options. One of them is comparing the R-Squared values for the models. With our models, only one of them is conducive to R-Squared comparisons. We can evaluate the MLR model this way, but we can only compare it to itself using different tuning methods. We can however find the R-Squared values for the correlations between the observed data and predicted data for each model.

```
knnRSQ <- cor(ben.test$`Rented Bike Count`, knnpredictions.en.tf) ** 2
dtRSQ <- cor(btf.test$`Rented Bike Count`, dtpredictions.tf) ** 2
mlrRSQ <- cor(btf.test$`Rented Bike Count`, mlrpredictions.tf) ** 2
```

The R-squared values of the models become lower from the kNN to Decision Tree to Multiple Regression models. They are 0.6586023, 0.65593 and 0.6209945. Based on these R-squared values, the kNN model performs the best.

The second method of evaluation I will use is Mean Absolute Deviation (MAD). This comparison is possible because it compares the absolute error between the predictions and true values.

```
knnMAD <- mean(abs(ben.test$`Rented Bike Count` - knnpredictions.en.tf))
dtMAD <- mean(abs(btf.test$`Rented Bike Count` - dtpredictions.tf))
mlrMAD <- mean(abs(btf.test$`Rented Bike Count` - mlrpredictions.tf))
```

The smaller the MAD, the more accurate the predictions. The kNN model had the lowest MAD at 0.1788698, with the decision tree second at 0.1888794, and MLR model last at a highest MAD of 0.2080769.

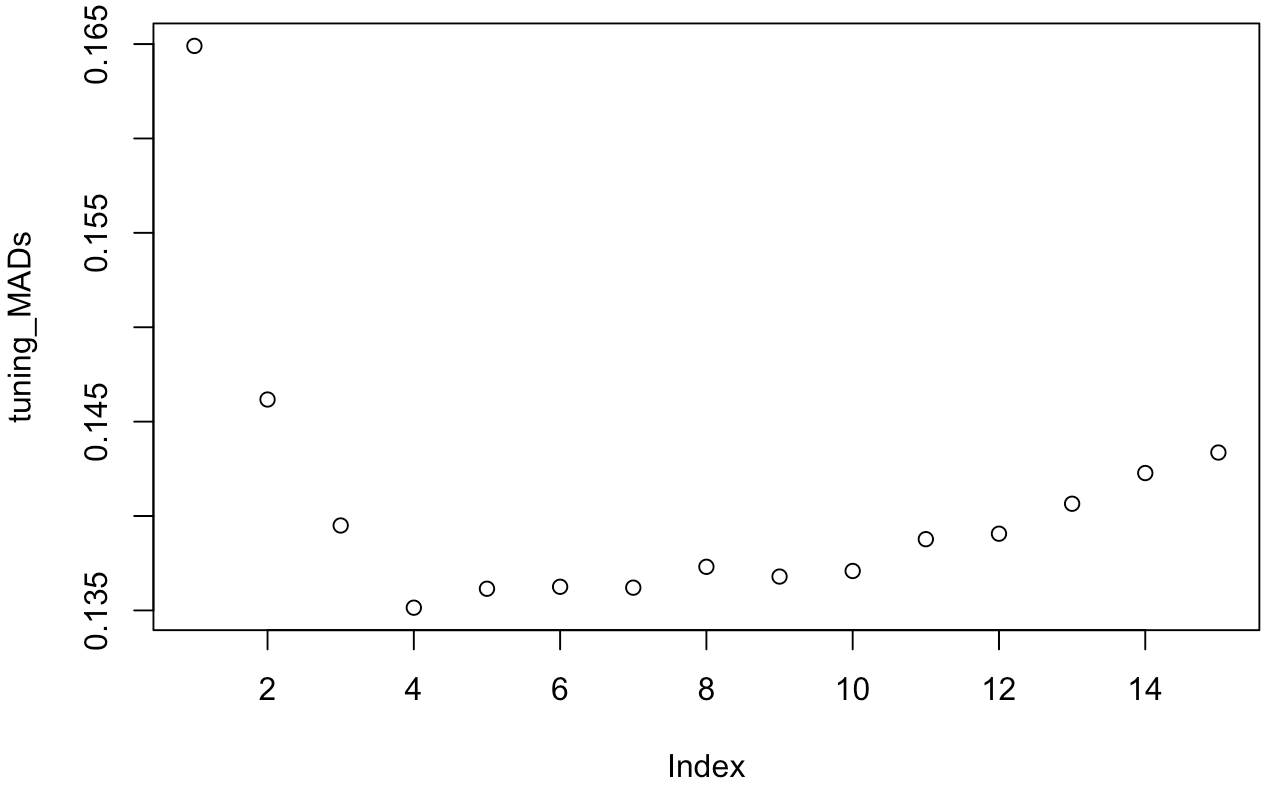
```
knnRMSE <- sqrt(mean((ben.test$`Rented Bike Count` - knnpredictions.en.tf) ** 2))
dtRMSE <- sqrt(mean((btf.test$`Rented Bike Count` - dtpredictions.tf) ** 2))
mlrRMSE <- sqrt(mean((btf.test$`Rented Bike Count` - mlrpredictions.tf) ** 2))
```

The smaller the Root Mean Squared Error (RMSE) the better the model. The RMSEs increase from the kNN model to the Decision Tree model to the Multiple Regression model. They are 0.2567413, 0.2513936, and 0.263557. This is the same finding as using the MAD.

Comparing the three models using R-squared, MAD, and RMSE metrics, the kNN model performs the best. But, the margins are small, so this order deviate even from small changes in the models. Some changes we can implement are using different k values and K-fold cross validation.

```
ks <- 15
tuning_MADs <- rep(0, ks)

for (i in 1:ks) {
  knnmodel_tuning <- knnreg(ben.train[,-2], ben.train[,2], k = i)
  knnpredictions_tuning <- predict(knnmodel_tuning, ben.test[,-2])
  tuning_MAD <- mean(abs(ben.test$`Rented Bike Count` - knnpredictions_tuning))
  tuning_MADs[i] <- tuning_MAD
}
plot(tuning_MADs)
```



For our kNN model, we only explored the use of one k value. Using other k values, we can see how the performance improves. Our initial k value was 81.7373843, which ended up being much too high. In the above code, we can see that the MAD bottoms out around k of five. This is a good estimate of performance, and we will continue using a k of 5 from now on.

```
# kNN
# Establish indexes for data splitting
enl <- nrow(bikedata.en)
random_ints <- sample(1:enl, size= enl, replace = FALSE)

# Establish list for MADs for each iteration of splits
knn_kfold_MADs <- rep(0, 5)

# Correct Date variable type
bikedata.en$Date <-  as.numeric(bikedata.en$Date) / 86400

# Running kNN with each permutation of data splits
for (i in 1:5) {
  if (i == 1) {
    inds <- random_ints[1:floor(.2*enl)]
  } else {
    lower <- (i - 1) * floor(.2*enl)
    upper <- (i) * floor(.2*enl)
    inds <- random_ints[lower:upper]
  }
  train <- bikedata.en[-inds,]
  test <- bikedata.en[inds,]
  model <- knnreg(train[,-2], train[,2], k = 5)
  preds <- predict(model, test[,-2])
  mad <- mean(abs(test[,2] - preds))
  knn_kfold_MADs[i] <- mad
}

# Find mean amount of MAD values across k-fold splits
mean_knn_MAD <- mean(knn_kfold_MADs)
```

To assess the models using k-fold cross validation, I made a loop to run the kNN model and find predictions for k=5 for neighbors and data splitting. The data was split into 80-20 percent sets for training and testing respectively, and then run to extract the MAD. I'm using the MAD because it is the easiest metric to understand, and it is easy to compare between sets of predictions. The mean MAD for the cross validation was 0.1372133.

```
# Decision Tree
# Get new randomizer seed to get new indexes
set.seed(23456)

# Establish indexes for data splitting
tfl <- nrow(bikedata.tf)
random_ints <- sample(1:tfl, size= tfl, replace = FALSE)

# Establish list for MADs for each iteration of splits
dt_kfold_MADs <- rep(0, 5)

# Running DT with each permutation of data splits
for (i in 1:5) {
  if (i == 1) {
    inds <- random_ints[1:floor(.2*tfl)]
  } else {
    lower <- (i - 1) * floor(.2*tfl)
    upper <- (i) * floor(.2*tfl)
    inds <- random_ints[lower:upper]
  }
  train <- bikedata.tf[-inds,]
  test <- bikedata.tf[inds,]
  model <- rpart(`Rented Bike Count` ~ ., data = train)
  preds <- predict(model, test[,-2])
  mad <- mean(unlist(abs(test[,2] - preds)))
  dt_kfold_MADs[i] <- mad
}

# Find mean amount of MAD values across k-fold splits
mean_dt_MAD <- mean(dt_kfold_MADs)

mean_dt_MAD
```

```
## [1] 0.1859047
```

With the same k-fold cross validation on the Decision Tree model, we get a mean MAD of 0.1859047.

```
# Multiple Regression
# Get new randomizer seed to get new indexes
set.seed(34567)

# Establish indexes for data splitting
tfl <- nrow(bikedata.tf)
random_ints <- sample(1:tfl, size= tfl, replace = FALSE)

# Establish list for MADs for each iteration of splits
mlr_kfold_MADs <- rep(0, 5)

# Running MLR with each permutation of data splits
for (i in 1:5) {
  if (i == 1) {
    inds <- random_ints[1:floor(.2*tfl)]
  } else {
    lower <- (i - 1) * floor(.2*tfl)
    upper <- (i) * floor(.2*tfl)
    inds <- random_ints[lower:upper]
  }
  train <- bikedata.tf[-inds,]
  test <- bikedata.tf[inds,]
  model <- lm(`Rented Bike Count` ~ ., data = train)
  preds <- predict(model, test[,-2])
  mad <- mean(unlist(abs(test[,2] - preds)))
  mlr_kfold_MADs[i] <- mad
}

# Find mean amount of MAD values across k-fold splits
mean_mlr_MAD <- mean(mlr_kfold_MADs)

mean_mlr_MAD
```

```
## [1] 0.2085872
```

And finally with our MLR model, we get a mean MAD of 0.2085872. Overall, the kNN model is still performing the best, with the Decision Tree second and the Multiple Regression model last. The kNN model has the lowest mean MAD and the multiple regression model has the highest. This is in line with our previous evaluations, and shows that the models didn't different due to sample selection bias.

Model Tuning and Performance Improvement

The models have some variation in their performance, but what if we combined the models into an ensemble? I am going to start by making an ensemble function that makes a prediction using each model and returns the average.

```
ensembleFunction <- function(encoded.train, encoded.test, train, test) {
  # kNN
  knnmodel.e <- knnreg(encoded.train[,-2], encoded.train[,2], k = 5)
  knnpreds.e <- predict(knnmodel.e, encoded.test[,-2])

  # DT
  dtmodel.e <- rpart(`Rented Bike Count` ~ ., data = train)
  dtpreds.e <- predict(dtmodel.e, test[,-2])

  # MLR
  mlrmodel.e <- lm(`Rented Bike Count` ~ ., data = train)
  mlrpreds.e <- predict(mlrmodel.e, test[,-2])

  avg.preds <- (knnpreds.e + dtpreds.e + mlrpreds.e) / 3

  return(avg.preds)
}
```

Now let's run the function so we can compare the predictions it makes.

```
e.predictions <- ensembleFunction(ben.train, ben.test, btf.train, btf.test)

# R-Square
eRSQ <- cor(btf.test$`Rented Bike Count`, e.predictions) ** 2
c(eRSQ, knnRSQ, dtRSQ, mlrRSQ)
```

```
## [1] 0.7835336 0.6586023 0.6559300 0.6209945
```

```
# MAD
eMAD <- mean(abs(btf.test$`Rented Bike Count` - e.predictions))
c(eMAD, knnMAD, dtMAD, mlrMAD)
```

```
## [1] 0.1540658 0.1788698 0.1888794 0.2080769
```

```
# RMSE
eRMSE <- sqrt(mean((btf.test$`Rented Bike Count` - e.predictions) ** 2))
c(eRMSE, knnRMSE, dtRMSE, mlrRMSE)
```

```
## [1] 0.2038669 0.2567413 0.2513936 0.2635570
```

After running the ensemble function to obtain predictions, we can evaluate it with the same metrics we used for each of the models separately. The R-squared value of the ensemble model to the observed values is 0.7835336, higher than any of the models alone, which means it is an improvement. The MAD of the ensemble is 0.1540658 and the RMSE is 0.2038669. These are both lower than any of the models alone, also indicating improvement. Based on these performance metrics, the ensemble function is a superior method than using any of the models alone.

For all of my evaluations, I calculated the R-squared, MAD, and RMSE for the models with the normalized and transformed values. This doesn't make the metrics any less useful, but it also doesn't show the real life number of Rented Bike Counts that are being predicted. To find the true values of the predictions, one would simply de-transform and de-normalize the data.

Another method of improving our model is bagging. I will use bagging on my decision tree model. The kNN model is considered relatively stable with low variance and won't benefit from re-sampling due to the nature of the model. The Multiple Linear Regression model is also low variance, and will not benefit much either.

```
bag <- bagging(
  formula = `Rented Bike Count` ~ .,
  data = btf.train,
  nbagg = 20,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0)
)

dtBagPred <- predict(bag, btf.test[,-2])

# R-square
bagRSQ <- cor(btf.test$`Rented Bike Count`, dtBagPred) ** 2
c(bagRSQ, dtRSQ)
```

```
## [1] 0.8496673 0.6559300
```

```
# MAD
bagMAD <- mean(abs(btf.test$`Rented Bike Count` - dtBagPred))
c(bagMAD, dtMAD)
```

```
## [1] 0.1093113 0.1888794
```

```
# RMSE
bagRMSE <- sqrt(mean((btf.test$`Rented Bike Count` - dtBagPred) ** 2))
c(bagRMSE, dtRMSE)
```

```
## [1] 0.1660789 0.2513936
```

When we recreate the Decision tree model, this time with bagging, we see that the model has improved. To make the bagged model, I used 20 iterations of bagging because of limitations set by my personal computer. The `coob` parameter is set to TRUE because I want to estimate the out of bag error as part of the refining of this model. The parameters `minsplit` and `cp` are 2 and 0 respectively so that the trees would be have a good balance between accuracy and comparability to the original DT model. After making the model and predictions, it performs better than the original (it has a higher R-squared, lower MAD and RMSE). This means bagging has improved the model.

Because bagging improved the model, we can incorporate the bagging model into our ensemble function, and see if the ensemble function improves as well.

```
ensembleFunction <- function(encoded.train, encoded.test, train, test) {
  # kNN
  knnmodel.e <- knnreg(encoded.train[,-2], encoded.train[,2], k = 5)
  knnpreds.e <- predict(knnmodel.e, encoded.test[,-2])

  # DT
  dtpreds.e <- predict(bag, test[,-2])

  # MLR
  mlrmodel.e <- lm(`Rented Bike Count` ~ ., data = train)
  mlrpreds.e <- predict(mlrmodel.e, test[,-2])

  avg.preds <- (knnpreds.e + dtpreds.e + mlrpreds.e) / 3

  return(avg.preds)
}

#------------------------------------------------------------#
# Compare
bag.e.predictions <- ensembleFunction(ben.train, ben.test, btf.train, btf.test)

# R-Square
bageRSQ <- cor(btf.test$`Rented Bike Count`, bag.e.predictions) ** 2
c(bageRSQ, eRSQ)
```

```
## [1] 0.8242403 0.7835336
```

```
# MAD
bageMAD <- mean(abs(btf.test$`Rented Bike Count` - bag.e.predictions))
c(bageMAD, eMAD)
```

```
## [1] 0.1327824 0.1540658
```

```
# RMSE
bageRMSE <- sqrt(mean((btf.test$`Rented Bike Count` - bag.e.predictions) ** 2))
c(bageRMSE, eRMSE)
```

```
## [1] 0.1824931 0.2038669
```

Using the new bagging model for the decision tree, we improve the ensemble function in all metrics. However, the bagging decision tree model actually performs better than the ensemble function. Changing the weights of the models in the ensemble function could improve the model yet.

```
w.ensembleFunction <- function(encoded.train, encoded.test, train, test) {
  # kNN
  knnmodel.e <- knnreg(encoded.train[,-2], encoded.train[,2], k = 5)
  knnpreds.e <- predict(knnmodel.e, encoded.test[,-2])

  # DT
  dtpreds.e <- predict(bag, test[,-2])

  # MLR
  mlrmodel.e <- lm(`Rented Bike Count` ~ ., data = train)
  mlrpreds.e <- predict(mlrmodel.e, test[,-2])

  avg.preds <- (knnpreds.e + 3 * dtpreds.e + mlrpreds.e) / 5

  return(avg.preds)
}

#------------------------------------------------------------#
# Compare
wbag.e.predictions <- w.ensembleFunction(ben.train, ben.test, btf.train, btf.test)

# R-Square
wbageRSQ <- cor(btf.test$`Rented Bike Count`, wbag.e.predictions) ** 2
c(wbageRSQ, bageRSQ, bagRSQ)
```

```
## [1] 0.8453930 0.8242403 0.8496673
```

```
# MAD
wbageMAD <- mean(abs(btf.test$`Rented Bike Count` - wbag.e.predictions))
c(wbageMAD, bageMAD, bagMAD)
```

```
## [1] 0.1191068 0.1327824 0.1093113
```

```
# RMSE
wbageRMSE <- sqrt(mean((btf.test$`Rented Bike Count` - wbag.e.predictions) ** 2))
c(wbageRMSE, bageRMSE, bagRMSE)
```

```
## [1] 0.1704444 0.1824931 0.1660789
```

Using an arbitrary weight of 3 for the bagging model of the decision tree within the ensemble model, we get a weighted bagged ensemble model that performs about as good as the bagging decision tree model. Perhaps with more tuning of the ensemble function, we could get it to eclipse the performance of the bagging DT model, but currently it performs well enough.

References

Seoul Bike Sharing Demand. (2020). UCI Machine Learning Repository. https://doi.org/10.24432/C5F62R

(https://doi.org/10.24432/C5F62R).