Bike sharing demand

Introduction

This document is created for the purpose of Bike Sharing Demand competition on Kaggle. The point of the competition is to predict the demand on the rented bikes based on the historical data.

```
train <- read.csv(file = "data/train.csv")
test <- read.csv(file = "data/test.csv")
```

There were 10886 training cases and 6493 test cases in the data.

Data Quality Assesment

The structure of the training set in R looks as follows:

```
str(train)
```

```
## 'data.frame':    10886 obs. of  12 variables:
##  $ datetime  : Factor w/ 10886 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
##  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
##  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
##  $ humidity  : int  81 80 80 75 75 75 80 86 75 76 ...
##  $ windspeed : num  0 0 0 0 0 ...
##  $ casual    : int  3 8 5 3 0 0 2 1 1 8 ...
##  $ registered: int  13 32 27 10 1 1 0 2 7 6 ...
##  $ count     : int  16 40 32 13 1 1 2 3 8 14 ...
```

Comparing this structure to feature description provided by Kaggle:

Data Fields

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

Data representation fixes

We can observe some problems with data representation in following variables:

- datetime
- season
- holiday
- workingday
- weather

Let's look at the data fields structure in more detail

```
summary(train)
```

```
##                 datetime         season         holiday       
##  2011-01-01 00:00:00:    1   Min.   :1.000   Min.   :0.00000  
##  2011-01-01 01:00:00:    1   1st Qu.:2.000   1st Qu.:0.00000  
##  2011-01-01 02:00:00:    1   Median :3.000   Median :0.00000  
##  2011-01-01 03:00:00:    1   Mean   :2.507   Mean   :0.02857  
##  2011-01-01 04:00:00:    1   3rd Qu.:4.000   3rd Qu.:0.00000  
##  2011-01-01 05:00:00:    1   Max.   :4.000   Max.   :1.00000  
##  (Other)            :10880                                    
##    workingday        weather           temp           atemp      
##  Min.   :0.0000   Min.   :1.000   Min.   : 0.82   Min.   : 0.76  
##  1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:13.94   1st Qu.:16.66  
##  Median :1.0000   Median :1.000   Median :20.50   Median :24.24  
##  Mean   :0.6809   Mean   :1.418   Mean   :20.23   Mean   :23.66  
##  3rd Qu.:1.0000   3rd Qu.:2.000   3rd Qu.:26.24   3rd Qu.:31.06  
##  Max.   :1.0000   Max.   :4.000   Max.   :41.00   Max.   :45.45  
##                                                                  
##     humidity        windspeed          casual         registered   
##  Min.   :  0.00   Min.   : 0.000   Min.   :  0.00   Min.   :  0.0  
##  1st Qu.: 47.00   1st Qu.: 7.002   1st Qu.:  4.00   1st Qu.: 36.0  
##  Median : 62.00   Median :12.998   Median : 17.00   Median :118.0  
##  Mean   : 61.89   Mean   :12.799   Mean   : 36.02   Mean   :155.6  
##  3rd Qu.: 77.00   3rd Qu.:16.998   3rd Qu.: 49.00   3rd Qu.:222.0  
##  Max.   :100.00   Max.   :56.997   Max.   :367.00   Max.   :886.0  
##                                                                    
##      count      
##  Min.   :  1.0  
##  1st Qu.: 42.0  
##  Median :145.0  
##  Mean   :191.6  
##  3rd Qu.:284.0  
##  Max.   :977.0  
## 
```

Below the proposed approach for different incorrectly represented data fields:

- **datetime** - variable is stored as Factor variable with 10886, so one level for every observations. We can solve this problem by extracting the time using some kind of string operations or using data handling packages. I will use R package *lubridate* for this purpose.
- **season** - this is 4 level factor variable, represented as a integer. The type change will be required for analysis
- **holiday** - this is binary variable, it might be interesting to connect this information with the day of the week extracted from datetime.
- **workingday** - another binary variable, should contain same information as day of the week excluding the holiday
- **weather** - factor variable with four levels represented as integer. Similar to the **season**. The commonsense would suggest that this variable is ordinal but results below does not support this assumption. While these results contradicting a little commonsense additional analysis should be performed.

```
print ('Average number of bikes rented by weather')
```

```
## [1] "Average number of bikes rented by weather"
```

```
tapply(train$count,INDEX = train$weather,mean)
```

```
##        1        2        3        4
## 205.2368 178.9555 118.8463 164.0000
```

The average number of bikes rented for **weather** category “4” which corresponds to really bad weather is higher then for category “3” which corresponds to slightly better weather. It could suggest the problems with the data or that snow alone is not real problem for cyclist. It would be interesting idea to confirm if this weather variable can describe the situation when snow is just on the ground and not snowing.

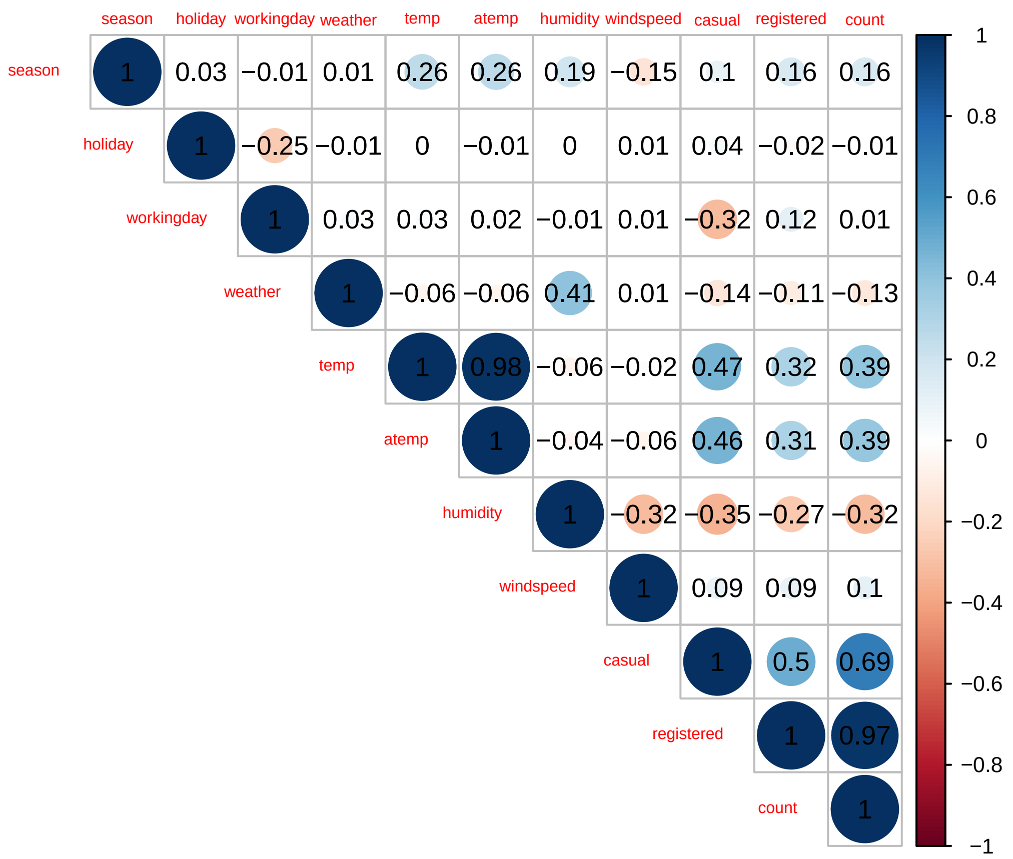
Correlation Analysis

The next step is to visualize correlation between variables to evaluate which of them could be removed before the model creation step is performed. The *corrplot* package is used for visualization, and the Pearson Correlation Coefficient was used for this case. Equation below:

$$\rho_{XY} = \frac{E[(X-\mu_X)](Y-\mu_Y)}{\sigma_X \sigma_Y}$$

where:
E is expectation
σ is standard deviation
μ is mean

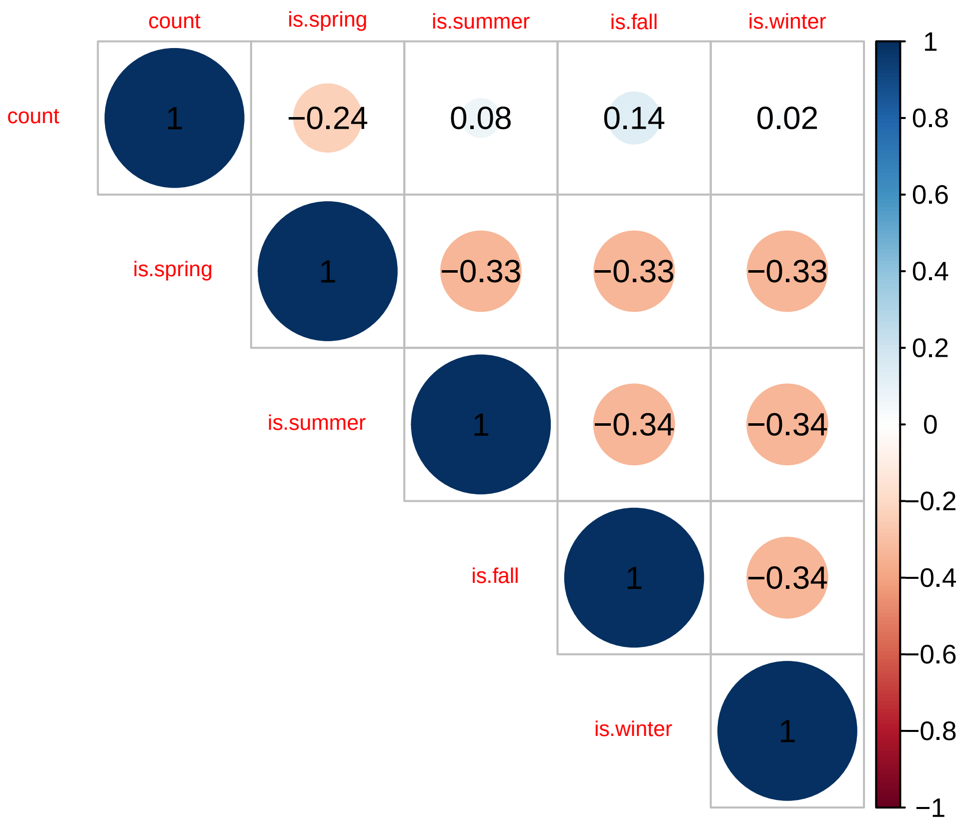
```
library(corrplot)
cor.train <- cor(train[,2:12])
par(cex = 0.8)
corrplot(cor.train, type = "upper",tl.srt=0,tl.cex = 0.65,tl.offset = 1,addCoef.col = "black")
```



The highest observed correlation is between **temp** and **atemp**. This lead us to conclusion that we should not include both variables in the model because they give the same information to the model. Additionally we can see that **casual** and **registered** variables are highly correlated to each other and to **count** but while these variables will not be used to model creation this is not particularly interesting.
The variables **atemp**,**temp** and **humidity** are strongly correlated to the **count** so they will be probably good predictors in the model.

The **season** variable is next on the list but while this is factor level variable and we store it as an integer some transformation must be done.

```
train$is.spring <- 0
train$is.spring[train$season == "1"] <- 1
train$is.summer <- 0
train$is.summer[train$season == "2"] <- 1
train$is.fall <- 0
train$is.fall[train$season == "3"] <- 1
train$is.winter <- 0
train$is.winter[train$season == "4"] <- 1
#sum(ctrain[,13]+ctrain[,14]+ctrain[,15]+ctrain[,16])
ctraincor <-cor(train[,12:16])
corrplot(ctraincor, type = "upper",tl.srt=0,tl.cex = 0.65,tl.offset = 1,addCoef.col = "black")
```



```r
rm(ctraincor)
```

From this figure we are interested only in first row because it contains information about correlation of number of bicycles rented with the season.

Cross-validation

For the purpose of the later model validation the sample of 10% of observations is excluded from the training set. This sample will be used later during model building exercise to compare how models behave on new data. We introduce it in case of overfitting of the model, and to provide first level of model verification. While this is not necessary for this case because we can use kaggle as a cross-validation engine or eventually use the source data to do model validation we keep this approach as a some sort of best practice. In the literature does not exist any particular fraction for sample, the proposed 10% is just proposed trade-off between the size of training data set and testing sample. To ensure reproductibility of this exercise we use *seed* parameter.

```r
set.seed(333)
index <- 1:nrow(train)
testindex <- sample(index,trunc(length(index)/10))
trainset <- train[-testindex,]
testset <- train[testindex,]
```

The new train set contains 9798 rows and test set 1088 observations.

Model selection

The first model I would like to present will be linear model based on least squares estimator. This model will be benchmark for further analysis.

Linear model

The least squares estimator is most commonly used estimator in statistical analysis. While the assumptions necessary for this estimator to be effective are in majority of the cases not meet it gives us good starting point for the further analysis. The model form:

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p$$

where:
Y - is column vector of outcomes
X - is input matrix and X_n is n-th column of this matrix

For the first linear model the features with reasonable correlation to the outcome will be selected. The next step will be including additional variables

Data preparation

During correlation analysis we realized that **temp**,**atemp** and **humidity** are good features for the first model. The additional variable that could be useful is season. We would use split of the season variable used during correlation analysis.

Estimation

Below the results of the first model.

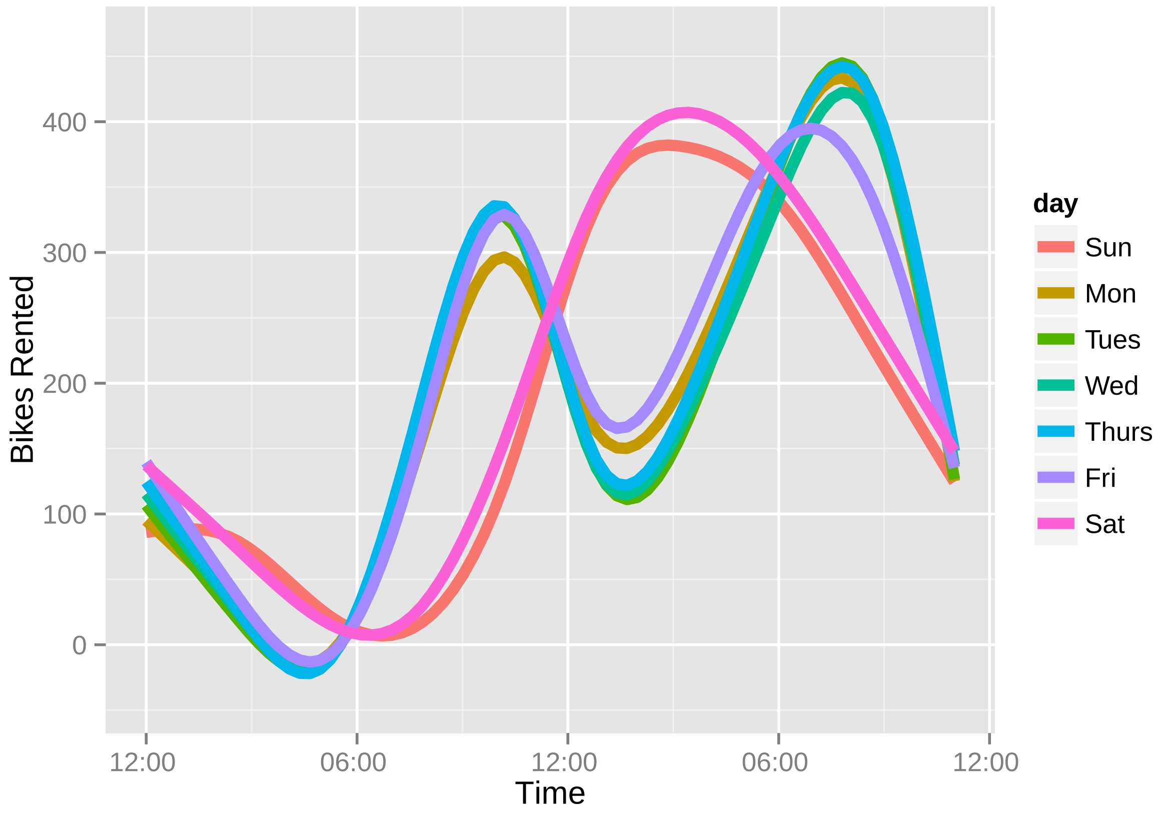
$$Y = \beta_0 + \beta_1 X_{atemp} + \beta_2 X_{humidity} + \beta_3 X_{is.spring} + \beta_4 X_{is.summer} + \beta_5 X_{is.fall}$$

```
lm.model.1 <-lm(trainset$count~trainset$atemp+trainset$humidity+trainset$is.spring+trainset$is.summer+t
summary(lm.model.1)
```

```
##
## Call:
## lm(formula = trainset$count ~ trainset$atemp + trainset$humidity +
##     trainset$is.spring + trainset$is.summer + trainset$is.fall)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -344.78 -102.22  -30.51   66.76  668.70
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        202.85047    9.37950   21.63   <2e-16 ***
## trainset$atemp       9.64201    0.29439   32.75   <2e-16 ***
## trainset$humidity   -2.97743    0.08443  -35.27   <2e-16 ***
## trainset$is.spring -65.49127    4.78007  -13.70   <2e-16 ***
## trainset$is.summer -63.50778    4.79468  -13.24   <2e-16 ***
## trainset$is.fall   -90.22425    5.71949  -15.78   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 154.7 on 9792 degrees of freedom
## Multiple R-squared:  0.2722, Adjusted R-squared:  0.2718
## F-statistic: 732.3 on 5 and 9792 DF,  p-value: < 2.2e-16
```

This is very basic model that does not include time information coded in **datetime** variable. We achieved R-squared at the level 27% which is not satisfying for us. So next step will be include hour to the model. Using *lubridate* we extracted hour and weekday and use this information to plot average number of bikes by weekday and hour. The additional packages used for this exercise were *ggplot2* and *scales*

```r
library(ggplot2)
library(scales)
library(lubridate)
train2 <- train
train2$day <- wday(ymd_hms(train2$datetime), label=TRUE)
train2$time <- as.POSIXct(strftime(ymd_hms(train2$datetime), format="%H:%M:%S"), format="%H:%M:%S")
ggplot(train2, aes(x=time, y=count, color=day)) +
  geom_smooth(fill = NA, size = 2) +
  xlab("Time") +
  ylab("Bikes Rented") +
  scale_x_datetime(labels=date_format("%I:%M %p"))
```



```r
rm(train2)
```

As we can see on above figure the number of Bikes rented is **not linearly** correlated to the time of the day. While we can keep using linear estimator it will be ineffective and we will drop this idea.

Results

The estimation of non-linear phenomena using linear estimator in this particular case most likely would be look like creating two different models. One for workdays and another for weekend+holidays because the structure of the bike rentals is different for these two cases. To achieve results we could use two another methods for each model. One would be creation of n-1 binary variables for every possible hour. From the point of view of statistics it would be calculation of conditional (on remaining variables) means for every possible hour. Second approach would be fitting Polynomial function (4rd grade for weekends and 6th grade for workday) and transforming the relation to linear.

Decision Trees

The decision tree model is second model proposed to solve this case. This model will be used as a benchmark for further analysis. We calculate decision tree using two packages *rpart* and *tree*. The decision trees model is basic model which can be enhanced by many different techniques of sampling or creating group of the decision trees.These methods will be presented later.

Data Preparation

The data preparation steep will be slightly easier for Random Forests because the input to the model can be in form of the factor variable.

In the opposite to linear model we will start with model with all possible data and then try to reduce number of features.

Estimation

```r
library(rpart)
library(Metrics)
dtmodel <- rpart(trainset$count~., data = extractFeatures(trainset))
submissiondt1 <- data.frame(datetime=testset$datetime, count=NA)
submissiondt1[, "count"] <- predict(dtmodel, extractFeatures(testset))
rmsle(actual = testset$count,predicted = submissiondt1$count)
```

```
## [1] 0.888838
```

```r
library(tree)
dtmodel2 <- tree(trainset$count~., data = extractFeatures(trainset))
submissiondt2 <- data.frame(datetime=testset$datetime, count=NA)
submissiondt2[, "count"] <- predict(dtmodel2, extractFeatures(testset))
rmsle(actual = testset$count,predicted = submissiondt2$count)
```

```
## [1] 0.9093533
```

The both models give similar results.

Random Forest

The last challenger from the category of Decision Trees will be Random Forests. Random forests are an ensemble learning method for classification or regression[like in this particular scenario]. Random forests works by construction a wide range of decision trees at training time and outputting mean prediction of the individual trees. Random forests advantage for decision trees is to reduce habit of overfitting on training set.

Estimation

We train model based on all variables and plot relative importance for the model.

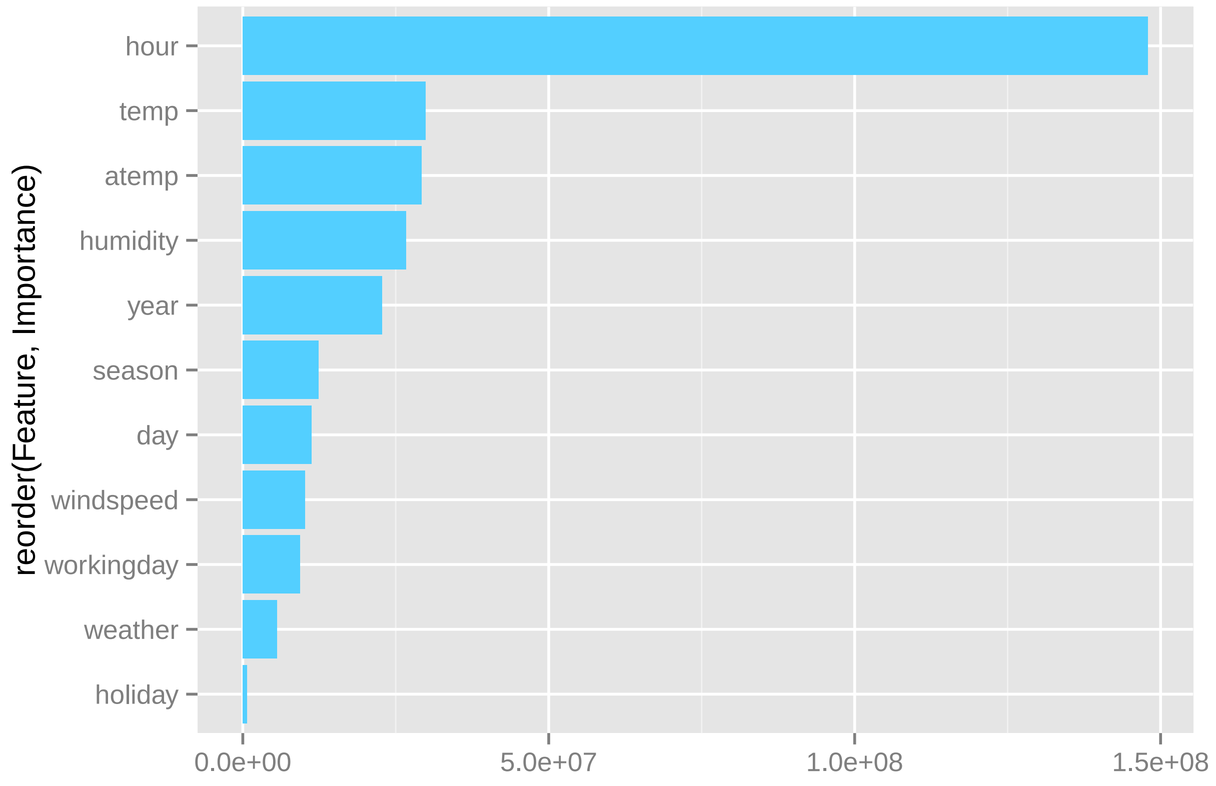
```
library(randomForest)
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
set1 <- extractFeatures(trainset)
rfmodel <- randomForest(set1,trainset$count, ntree=100)
summary(rfmodel)
```

```
##                 Length Class  Mode
## call               4   -none- call
## type               1   -none- character
## predicted       9798   -none- numeric
## mse              100   -none- numeric
## rsq              100   -none- numeric
## oob.times       9798   -none- numeric
## importance        11   -none- numeric
## importanceSD       0   -none- NULL
## localImportance    0   -none- NULL
## proximity          0   -none- NULL
## ntree              1   -none- numeric
## mtry               1   -none- numeric
## forest            11   -none- list
## coefs              0   -none- NULL
## y               9798   -none- numeric
## test               0   -none- NULL
## inbag              0   -none- NULL
```

```
rf1imp <- importance(rfmodel)
rf1fi <- data.frame(Feature=row.names(rf1imp), Importance=rf1imp[,1])
ggplot(rf1fi, aes(x=reorder(Feature, Importance), y=Importance)) +
    geom_bar(stat="identity", fill="#53cfff") + coord_flip() +ylab("")
```

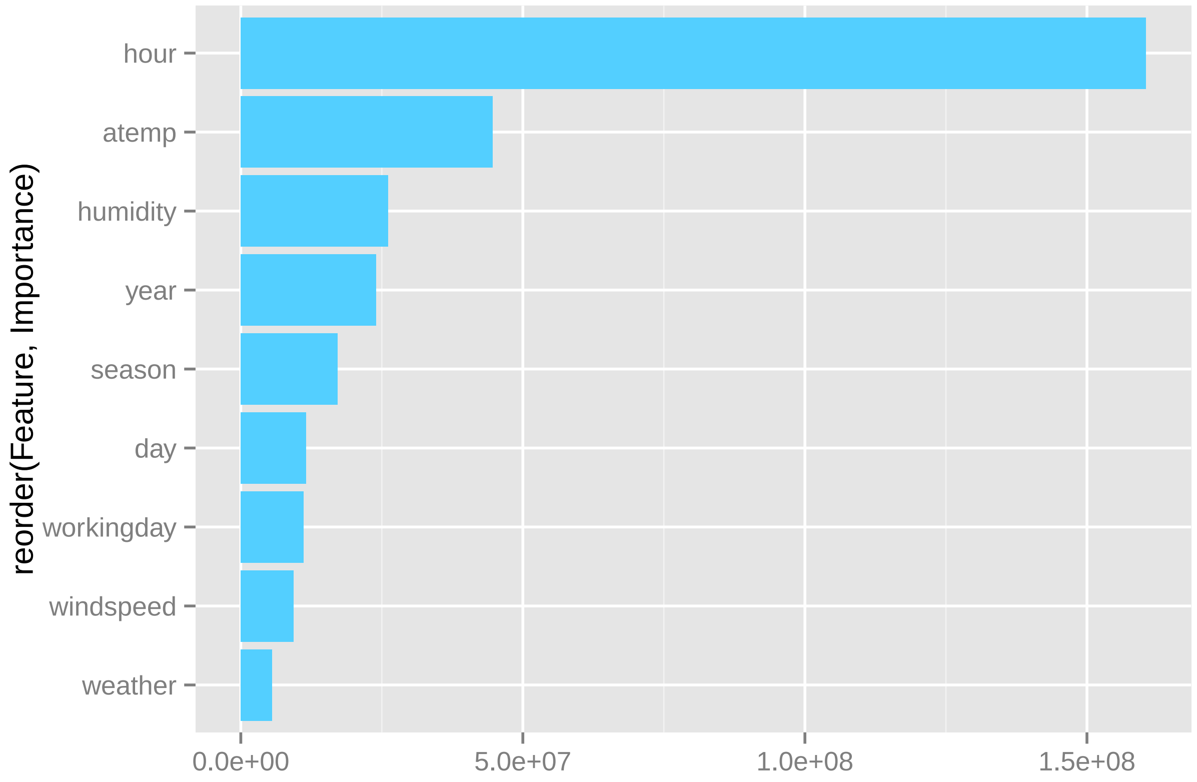


We can observe that **holiday** feature is not useful from the point of this model, additionally we would like to remove temp while it is strictly correlated to **atemp** and both variables contain same information. So we create second version of the model without these two variables.

```
set2 <- extractFeatures(trainset)
rfmodel2 <- randomForest(set2,trainset$count, ntrees = 100)
summary(rfmodel2)
```

```
##                 Length Class  Mode
## call               4   -none- call
## type               1   -none- character
## predicted       9798   -none- numeric
## mse              500   -none- numeric
## rsq              500   -none- numeric
## oob.times       9798   -none- numeric
## importance         9   -none- numeric
## importanceSD       0   -none- NULL
## localImportance    0   -none- NULL
## proximity          0   -none- NULL
## ntree              1   -none- numeric
## mtry               1   -none- numeric
## forest            11   -none- list
## coefs              0   -none- NULL
## y               9798   -none- numeric
## test               0   -none- NULL
## inbag              0   -none- NULL
```

```
rf2imp <- importance(rfmodel2)
rf2fi <- data.frame(Feature=row.names(rf2imp), Importance=rf2imp[,1])
ggplot(rf2fi, aes(x=reorder(Feature, Importance), y=Importance)) +
    geom_bar(stat="identity", fill="#53cfff") + coord_flip() +ylab("")
```



Now we can create submission to Kaggle.

```
submission1 <- data.frame(datetime=testset$datetime, count=NA)
submission1[, "count"] <- predict(rfmodel2, extractFeatures(testset))
#write.csv(submission, file = "2_random_forest_submission.csv", row.names=FALSE)
```

Alternative mode

While the random forest generated on all data is interesting option. The model would be more useful from the point of described problem if the source data will be only from current month. Below the model which creates random forest for every month separately

```
extractFeatures <- function(data) {
  features <- c("season",
                "workingday",
                "weather",
                "atemp",
                "humidity",
                "windspeed",
                "hour",
                "day")
  data$hour <- hour(ymd_hms(data$datetime))
  data$day <- wday(ymd_hms(data$datetime), label=TRUE)
  return(data[,features])
}
trainFea <- extractFeatures(trainset)
testFea  <- extractFeatures(testset)
submission2 <- data.frame(datetime=testset$datetime, count=NA)
for (i_year in unique(year(ymd_hms(testset$datetime)))) {
  for (i_month in unique(month(ymd_hms(testset$datetime)))) {
    testLocs   <- year(ymd_hms(testset$datetime))==i_year & month(ymd_hms(testset$datetime))==i_month
    testSubset <- testset[testLocs,]
    trainLocs  <- ymd_hms(trainset$datetime) <= min(ymd_hms(testSubset$datetime))
    rf <- randomForest(extractFeatures(trainset[trainLocs,]), trainset[trainLocs,"count"], ntree=100)
    submission2[testLocs, "count"] <- predict(rf, extractFeatures(testSubset))
  }
}
```

Results

The two approaches were proposed. First one was based on linear estimator. But while the data structure suggested that independent variables does not explain the dependent variable in linear way the random forest models were proposed. From the two Random Forest models one was created based on the whole data set and second was set of different random forests models for each month separately. The difference between models were not significant so second approach sounds more reliable.

Evaluation metrics

The results are evaluated based on Root Mean Squared Logarithmic Error (RMSLE) which is calculated as follows

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(p_i+1)-log(a_i+1))^2}$$

where:

- n is number of hours in the test set
- p_i is predicted count
- a_i is the actual count
- log(x) is the natural logarithm

To verify the accuracy of the model the implementation of this mechanism should be created in R. Fortunately this function is implemented in package *metrics*

Model Name	RMSLE
Decision Tree 1	0.888838
Decision Tree 2	0.9093533
Random Forests 1	0.5159354

Model Name	RMSLE
Random Forests 2	0.4378783
Random Forests 3	0.697618

Final

From the point of the view of RMSLE Random Forest 2 looks like the best solution. Unfortunately it allowed me obtain position 1922 for around 3000 teams. Next steps to improve the performance of the model would be as follows:

- try bagging and boosting
- plot the test and training error to verify are we overfit or underfit
- play with variables and script parameters.