HarvardX: PH125.9x Data Science
Temperature Forecasting

Sakshi Gupta

22/12/2020

Contents

1 Introduction

This is the report on Temperature Forecast.This is the capstone report of the HarvardX: Data Science-Capstone course project 2.

2 Objective

In this project, We have to predict the maximum temperature of the last day of the year. We have the temperature historical data of the whole year which will be used in prediction. This is machine learning problem, using supervised learning. Supervised learning can be described as taking an input vector comprised of n-features and mapping it to an associated target value or class label.

3 Data

The original data was obtained from National Centers for Environmental Prediction and can be found at https://www.ncep.noaa.gov/

3.1 Loading of Data

The data can be loaded by the below code. We will split the complete data into train set and test set. we will train our model on the train set and test it on the test set

```
##########################################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Warning: package 'tidyverse' was built under R version 4.0.3

## Warning: package 'ggplot2' was built under R version 4.0.3

## Warning: package 'readr' was built under R version 4.0.3

## Warning: package 'stringr' was built under R version 4.0.3
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("randomForest", repos = "http://cran.us.r-project.org")


#loading libraries
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(dplyr)
library(randomForest)
```

```r
dates<- read.csv('temps_new.csv' , stringsAsFactors = TRUE)
str(dates)
```

```
## 'data.frame':    348 obs. of  9 variables:
##  $ year   : int  2019 2019 2019 2019 2019 2019 2019 2019 2019 2019 ...
##  $ month  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ week   : Factor w/ 7 levels "Fri","Mon","Sat",..: 1 3 4 2 6 7 5 1 3 4 ...
##  $ temp_2 : int  45 44 45 44 41 40 44 51 45 48 ...
##  $ temp_1 : int  45 45 44 41 40 44 51 45 48 50 ...
##  $ average: num  45.6 45.7 45.8 45.9 46 46.1 46.2 46.3 46.4 46.5 ...
##  $ actual : int  45 44 41 40 44 51 45 48 50 52 ...
##  $ friend : int  29 61 56 53 41 40 38 34 47 49 ...
```

Data Columns/ Attributes
1. Year : 2019
2. Month : Number for month of the year
3. Day : Number for day of the year
4. week : Day of the week as a chracter string
5. temp_2 : Max. Temperature 2 days prior
6. temp_1 : Max. Temperature 1 days prior
7. average : Historical average max temperature
8. actual : Actual Max temperature
9. Friend : Friend's prediction, a random number between 20 below the average and 20 above the average

4 Data Preperation

If we observe the data we will see that there are total 348 rows whereas in a year we have 365 days. That means we have less data as expected, but since this is not a big number,missing data will not have large effect. Also, this data is from a very trusted source we can say that the data quality is good. The data has 9 columns with 8 features and one target - "actual"

5 Exploratory Data Analysis

5.1 Data Pre-processing

Preprocessing steps

1. One-hot coding
2. split data - into features and labels
3. Split data into training and tests sets

Identify anomalies in each column of the dataset using Summary

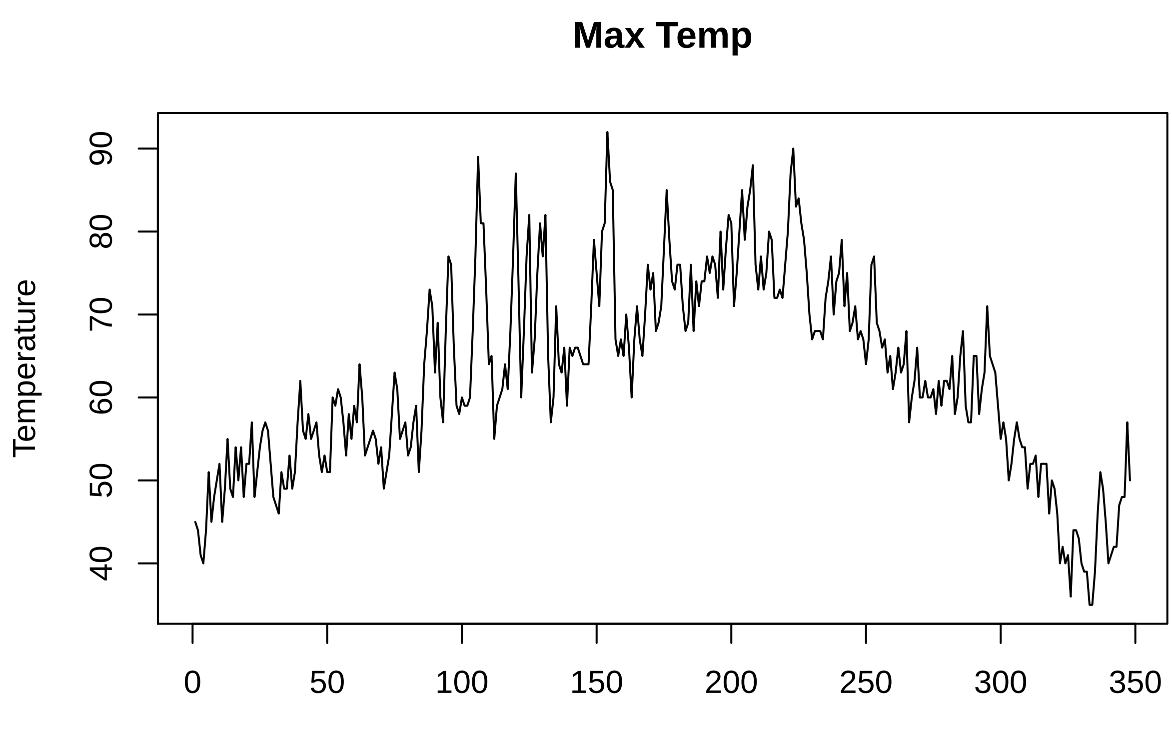
```r
summary(dates)
```

```
##       year          month             day            week        temp_2
##  Min.   :2019   Min.   : 1.000   Min.   : 1.00   Fri  :50   Min.   : 35.00
```

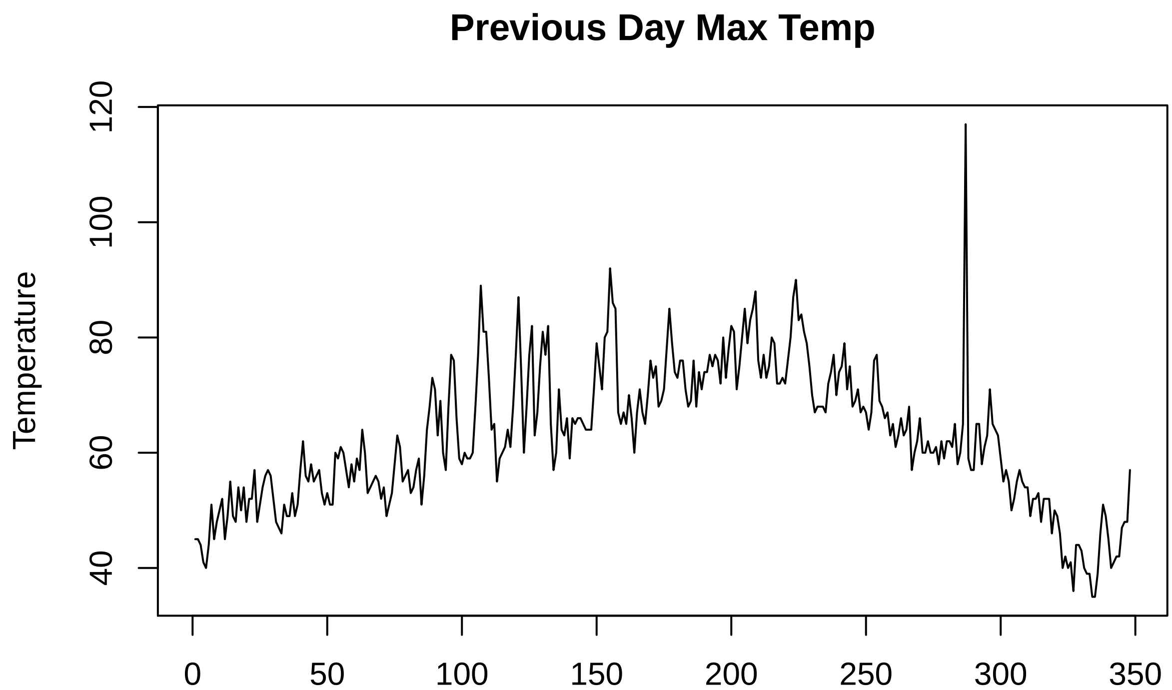
```
##  1st Qu.:2019   1st Qu.: 3.000   1st Qu.: 8.00   Mon   :49   1st Qu.: 54.00
##  Median :2019   Median : 6.000   Median :15.00   Sat   :50   Median : 62.50
##  Mean   :2019   Mean   : 6.477   Mean   :15.51   Sun   :49   Mean   : 62.65
##  3rd Qu.:2019   3rd Qu.:10.000   3rd Qu.:23.00   Thurs:49   3rd Qu.: 71.00
##  Max.   :2019   Max.   :12.000   Max.   :31.00   Tues :52   Max.   :117.00
##                                                  Wed  :49
##      temp_1          average          actual          friend
##  Min.   : 35.0   Min.   :45.10   Min.   :35.00   Min.   :28.00
##  1st Qu.: 54.0   1st Qu.:49.98   1st Qu.:54.00   1st Qu.:47.75
##  Median : 62.5   Median :58.20   Median :62.50   Median :60.00
##  Mean   : 62.7   Mean   :59.76   Mean   :62.57   Mean   :60.03
##  3rd Qu.: 71.0   3rd Qu.:69.03   3rd Qu.:71.00   3rd Qu.:71.00
##  Max.   :117.0   Max.   :77.40   Max.   :92.00   Max.   :95.00
##
```

Just by looking at the summary of the data, it becomes difficult to find out any anomalities. But by using the graphs, any anomalities looks clearly.

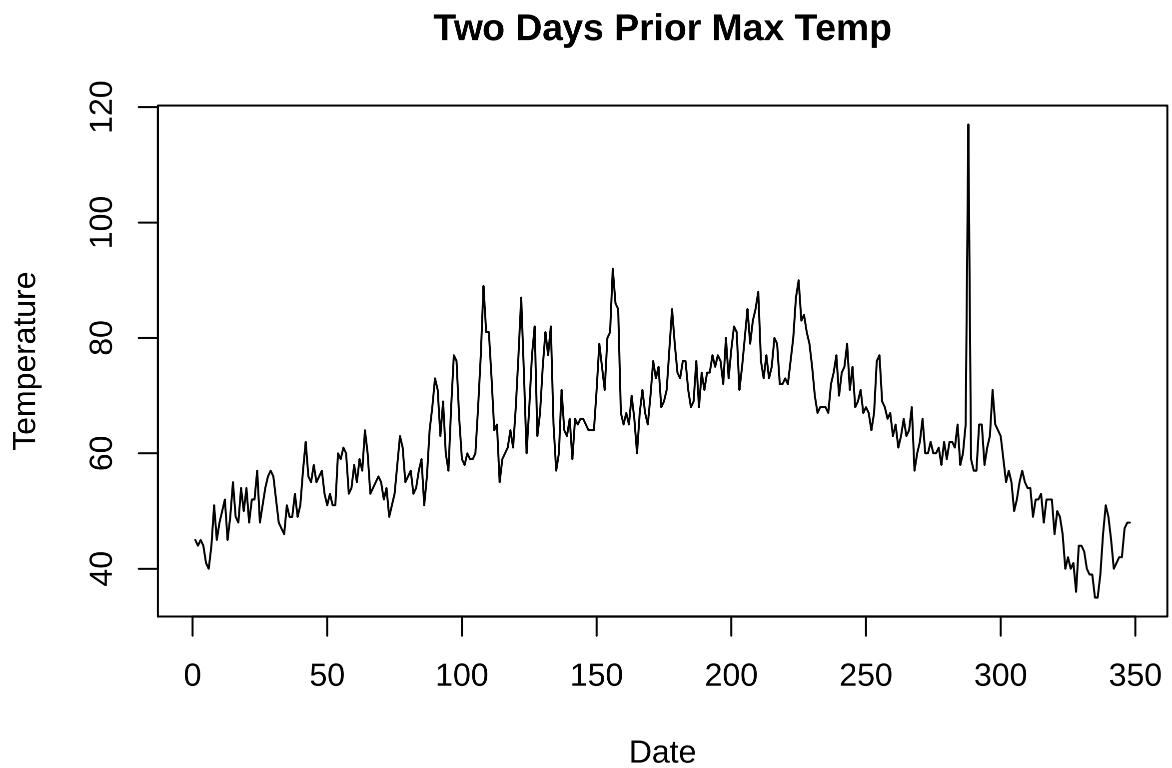
```r
plot(dates$actual, type = "l", ylab = "Temperature", xlab = " ", main = "Max Temp")
```



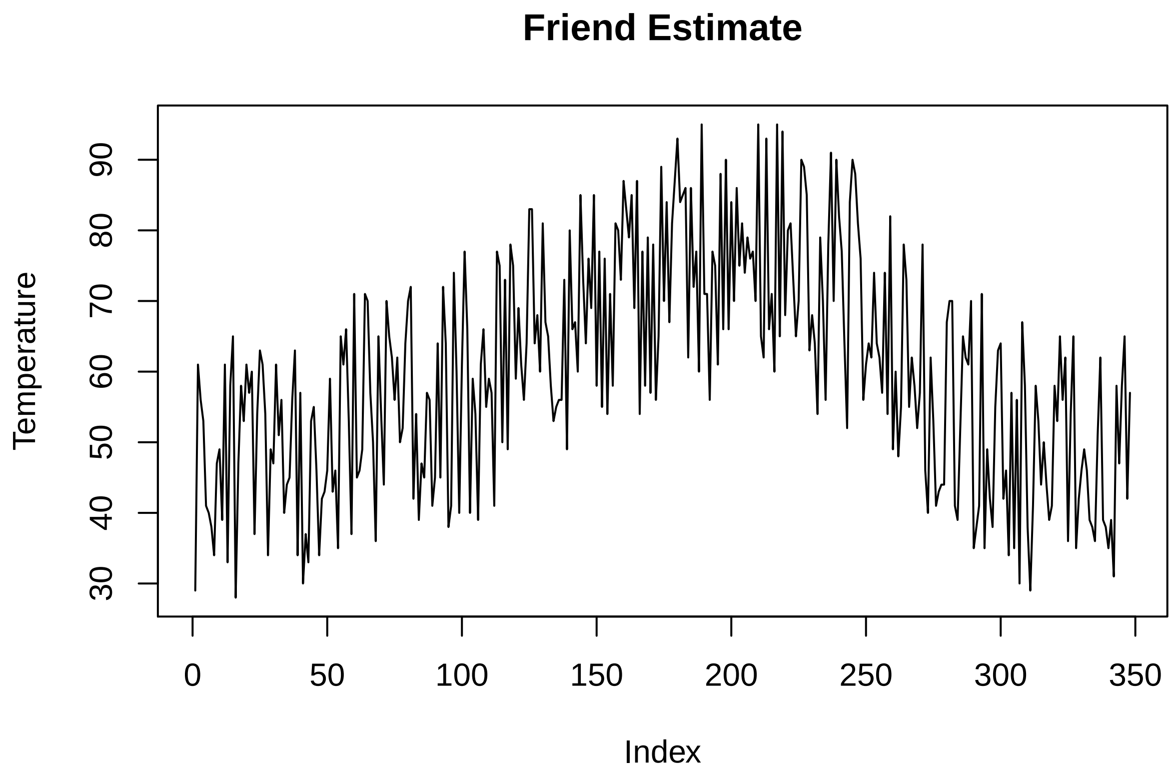

```r
plot(dates$temp_1, type = "l", ylab = "Temperature", xlab = " ", main= "Previous Day Max Temp")
```




```
plot(dates$temp_2, type = "l", ylab = 'Temperature', xlab = 'Date',  main= "Two Days Prior Max Temp")
```




```r
plot(dates$friend, type = "l", ylab = "Temperature", main= "Friend Estimate")
```



Let's convert the seperated dates to single date.

```
years <- dates$year
months <- dates$month
day <- dates$day
```

Now lets convert them to date format now.

```
date <- dates %>%
  mutate(date = make_date(year, month, day))
```

Lets start pre-processing the data

5.1.1 1. one hot coding - using dummyvars of caret package

What is one-hot coding? It takes input as categorical data and converts them to the numerical data without any ordering

```
dmy <- dummyVars(" ~ .", data = dates)
newdates <- data.frame(predict(dmy, newdata = dates))
glimpse(newdates)
```

```
## Rows: 348
## Columns: 15
```

```
## $ year       <dbl> 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 20...
## $ month      <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ day        <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1...
## $ week.Fri   <dbl> 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,...
## $ week.Mon   <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,...
## $ week.Sat   <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,...
## $ week.Sun   <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,...
## $ week.Thurs <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,...
## $ week.Tues  <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,...
## $ week.Wed   <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,...
## $ temp_2     <dbl> 45, 44, 45, 44, 41, 40, 44, 51, 45, 48, 50, 52, 45, 49, ...
## $ temp_1     <dbl> 45, 45, 44, 41, 40, 44, 51, 45, 48, 50, 52, 45, 49, 55, ...
## $ average    <dbl> 45.6, 45.7, 45.8, 45.9, 46.0, 46.1, 46.2, 46.3, 46.4, 46...
## $ actual     <dbl> 45, 44, 41, 40, 44, 51, 45, 48, 50, 52, 45, 49, 55, 49, ...
## $ friend     <dbl> 29, 61, 56, 53, 41, 40, 38, 34, 47, 49, 39, 61, 33, 58, ...
```

5.1.2 2. Split dataframe into features and target

Features are the columns used to make predctions and labels are the target which we have to predict

```
features <- newdates %>% select('year','month','day','week.Fri','week.Mon','week.Sat','week.Sun','week.'
target <- newdates %>% select('actual')
```

5.1.3 3. Split data into train and test set

We will split the data into train and test set and will use the train set to train the model and test set will be used to validate the model.

```
set.seed(1, sample.kind = "Rounding")  # if using R 3.5 or earlier, use `set.seed(1)`
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y= newdates$actual, times = 1, p = 0.25, list = FALSE)

train_set <- newdates[-test_index,]
test_set <- newdates[test_index,]
```

Analyse the train set and test set

```
summary(train_set)
```

```
##       year          month             day           week.Fri
##  Min.   :2019   Min.   : 1.000   Min.   : 1.00   Min.   :0.0000
##  1st Qu.:2019   1st Qu.: 3.000   1st Qu.: 7.00   1st Qu.:0.0000
##  Median :2019   Median : 7.000   Median :14.00   Median :0.0000
##  Mean   :2019   Mean   : 6.523   Mean   :14.88   Mean   :0.1615
##  3rd Qu.:2019   3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:0.0000
##  Max.   :2019   Max.   :12.000   Max.   :31.00   Max.   :1.0000
##     week.Mon         week.Sat         week.Sun        week.Thurs
```

```
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00  
##  Median :0.0000   Median :0.0000   Median :0.0000   Median :0.00  
##  Mean   :0.1154   Mean   :0.1385   Mean   :0.1577   Mean   :0.15  
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00  
##    week.Tues         week.Wed          temp_2          temp_1      
##  Min.   :0.0000   Min.   :0.0000   Min.   : 35.0   Min.   : 35.00  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 54.0   1st Qu.: 54.75  
##  Median :0.0000   Median :0.0000   Median : 62.0   Median : 61.00  
##  Mean   :0.1346   Mean   :0.1423   Mean   : 62.5   Mean   : 62.53  
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.: 71.0   3rd Qu.: 71.00  
##  Max.   :1.0000   Max.   :1.0000   Max.   :117.0   Max.   :117.00  
##     average          actual         friend     
##  Min.   :45.10   Min.   :35.0   Min.   :29.00  
##  1st Qu.:49.98   1st Qu.:54.0   1st Qu.:47.00  
##  Median :57.65   Median :62.5   Median :58.50  
##  Mean   :59.45   Mean   :62.4   Mean   :59.45  
##  3rd Qu.:68.72   3rd Qu.:71.0   3rd Qu.:70.00  
##  Max.   :77.40   Max.   :92.0   Max.   :95.00
```

```r
summary(test_set)
```

```
##       year          month             day           week.Fri      
##  Min.   :2019   Min.   : 1.000   Min.   : 2.00   Min.   :0.00000  
##  1st Qu.:2019   1st Qu.: 4.000   1st Qu.:11.00   1st Qu.:0.00000  
##  Median :2019   Median : 6.000   Median :18.00   Median :0.00000  
##  Mean   :2019   Mean   : 6.341   Mean   :17.38   Mean   :0.09091  
##  3rd Qu.:2019   3rd Qu.: 9.000   3rd Qu.:24.00   3rd Qu.:0.00000  
##  Max.   :2019   Max.   :12.000   Max.   :31.00   Max.   :1.00000  
##     week.Mon         week.Sat         week.Sun         week.Thurs    
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.0000  
##  Median :0.0000   Median :0.0000   Median :0.00000   Median :0.0000  
##  Mean   :0.2159   Mean   :0.1591   Mean   :0.09091   Mean   :0.1136  
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.0000  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000  
##    week.Tues         week.Wed          temp_2          temp_1     
##  Min.   :0.0000   Min.   :0.0000   Min.   :35.00   Min.   :35.00  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:54.00   1st Qu.:53.75  
##  Median :0.0000   Median :0.0000   Median :63.50   Median :64.50  
##  Mean   :0.1932   Mean   :0.1364   Mean   :63.11   Mean   :63.19  
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:72.00   3rd Qu.:71.25  
##  Max.   :1.0000   Max.   :1.0000   Max.   :90.00   Max.   :92.00  
##     average          actual          friend     
##  Min.   :45.10   Min.   :35.00   Min.   :28.00  
##  1st Qu.:50.05   1st Qu.:54.00   1st Qu.:49.75  
##  Median :60.95   Median :62.50   Median :62.00  
##  Mean   :60.67   Mean   :63.08   Mean   :61.76  
##  3rd Qu.:69.85   3rd Qu.:71.00   3rd Qu.:73.25  
##  Max.   :77.40   Max.   :90.00   Max.   :95.00
```

Lets check the number of days in each month in the dataset

```r
newdates %>% group_by(month) %>% summarise(n = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 12 x 2
##    month     n
##    <dbl> <int>
##  1     1    31
##  2     2    26
##  3     3    31
##  4     4    30
##  5     5    31
##  6     6    30
##  7     7    31
##  8     8    19
##  9     9    28
## 10    10    30
## 11    11    30
## 12    12    31
```

Check any NA value in the dataset

```r
anyNA(train_set)
```

```
## [1] FALSE
```

```r
anyNA(test_set)
```

```
## [1] FALSE
```

Fetch all the column names except target column name 'actual' in the list for using them in future

```r
featuresCols <- colnames(newdates[-14])
```

6 Model Building

Lets calculate the base model using the result of which we can see how can we improve our further models

6.1 1. Simple Average Model

```r
base_avg_values_test <- test_set[ , "average"]
base_actual_values_test <- test_set[ , "actual"]


base_err <- abs(base_avg_values_test  - base_actual_values_test)
#calculate the average base error in degrees
mean_avg <- mean(base_err)
mean_avg
```

```
## [1] 4.515909
```

```
#calculate the MAPE -Mean absolute percentage error
mape_avg = 100 * (mean_avg / base_actual_values_test)

#accuracy
accuracy_avg = 100 - mean(mape_avg)
accuracy_avg
```

```
## [1] 92.57712
```

Lets save this observed values in a table and keep on adding more to this table to analyse the best model

```
predicted_results <- data_frame(Date = "31-12-2019" , Method = "Average Model appraoch",  Error = mean_a
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
predicted_results %>% knitr::kable()
```

Date	Method	Error	Accuracy
31-12-2019	Average Model appraoch	4.515909	92.57712

With this avg we get the forecast with an error of 4.51 degrees and accuracy of 92.4%. This is our baseline error and now we have to build such model which will give us less error than this base error

Now lets train our model using

6.2 2. RANDOM FOREST

Random forest is an ensemble-based learning algorithm which is comprised of n collections of de-correlated decision trees [10]. It is built off the idea of bootstrap aggregation, which is a method for resampling with replacement in order to reduce variance.Random Forest uses multiple trees to average for regression in the terminal leaf nodes when making a prediction.

Because of the idea of decision trees, random forest models have resulted in significant improvements in prediction accuracy as compared to other models

```
set.seed(22, sample.kind = 'Rounding')
```

```
## Warning in set.seed(22, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
temp.rf <- randomForest(actual ~ . , data = train_set,
                        importance = TRUE, na.action = na.omit)

temp.rf
```

```
##
## Call:
##  randomForest(formula = actual ~ ., data = train_set, importance = TRUE,      na.action = na.omit)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 21.94584
##                     % Var explained: 84.14
```

Lets check the predictions

```
pred = predict(temp.rf, newdata = test_set)
```

Now lets calculate the absolute error between the predicted values and the actual value

```
abs_erre_rf <- abs(pred - base_actual_values_test)
```

take the mean of absolute error

```
abs_mean_rf <- mean(abs_erre_rf)
abs_mean_rf
```

```
## [1] 4.190482
```

```
#calculate MAPE- Mean absolute percentage error
mape = 100 * (abs_erre_rf / base_actual_values_test)


#accuracy
accuracy = 100 - mean(mape)
accuracy
```

```
## [1] 93.26397
```

Adding this to the results table

```
predicted_results <- bind_rows(predicted_results,data_frame(Date = "31-12-2019" , Method = "Random_fore
predicted_results %>% knitr::kable()
```

Date	Method	Error	Accuracy
31-12-2019	Average Model appraoch	4.515909	92.57712
31-12-2019	Random_forest	4.190482	93.26397

Here we observed that the new model has improved the error.

Now we will calculate MAPE- Mean absolute percentage error

```r
mape = 100 * (abs_erre_rf / base_actual_values_test)
```

Lets calculate the accuracy of this model

```r
accuracy = 100 - mean(mape)
accuracy
```

```
## [1] 93.26397
```

Lets work on improving this model by randomly tunning some parameters

```r
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
temp.rf_new <- randomForest(actual ~ . , data = train_set, mtry=4 ,ntree = 120,
                        importance = TRUE, na.action = na.omit)

temp.rf_new
```

```
##
## Call:
##  randomForest(formula = actual ~ ., data = train_set, mtry = 4,      ntree = 120, importance = TRUE,
##                Type of random forest: regression
##                      Number of trees: 120
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 21.95478
##                     % Var explained: 84.13
```

Lets check the predictions for this tuned model

```r
pred_new = predict(temp.rf_new, newdata = test_set)
```

now lets calculate the absolute error between the predicted values and the actual value

```r
abs_erre_rf_new <- abs(pred_new - base_actual_values_test)
```

take the mean of absolute error

```r
abs_mean_rf_new <- mean(abs_erre_rf_new)
abs_mean_rf_new
```

```
## [1] 4.148032
```

```
#calculate MAPE- Mean absolute percentage error
mape_new = 100 * (abs_erre_rf_new / base_actual_values_test)


#accuracy
accuracy_new = 100 - mean(mape_new)
accuracy_new
```

```
## [1] 93.31148
```

Adding these values also to the results table

```
predicted_results <- bind_rows(predicted_results, data_frame(Date = "31-12-2019" , Method = "Random_fore

predicted_results %>% knitr::kable()
```

Date	Method	Error	Accuracy
31-12-2019	Average Model appraoch	4.515909	92.57712
31-12-2019	Random_forest	4.190482	93.26397
31-12-2019	Random_forest_improved	4.148032	93.31148

Here we observed that the new model has further improved the error. Similarly we can tune other hyperparameters as well

7 Variable importance

7.1 Variable importance and feature selection

By now we have made a good model which is giving us improved error values. Now on observing, we can see that there are many features which are not contributing much in the prediction of the temperature. Here we need to identify which are the top features which are contributing in the better prediction and making the model much improved

```
importance(temp.rf)
```

```
##               %IncMSE IncNodePurity
## year         0.000000       0.00000
## month       15.054191    3018.55458
## day          4.471118     847.42757
## week.Fri    -2.678798     140.24480
## week.Mon    -2.417115     144.25167
## week.Sat    -3.515902      92.60672
## week.Sun    -3.844360     196.82099
## week.Thurs -3.975306      84.96002
## week.Tues    2.746024     169.94256
## week.Wed    -3.020290      69.01941
## temp_2      15.238167    6938.42919
## temp_1      28.638841   11593.42726
```

```
## average    22.866353    9694.23254
## friend     10.917735    1876.92098
```

we see received two paramenters - %IncMSE & IncNodePurity

Mean Decrease Accuracy (%IncMSE) - This shows how much our model accuracy decreases if we leave out that variable.The higher number, the more important the feature is.

IncNodePurity - This is a measure of variable importance based on the Gini impurity index used for the calculating the splits in trees.

So the two important features are 1.temp_1- the maximum temperature the day before 2.average - the historical avg max temperature

The day of the week and the year are not contributing much in predicting the max temperature as it has nothing to do with the temperature To improve our model we will only consider these two paramenters and calculate the error

7.2 Model only with important features

Here we can remove all other variables which has no/less importance and build the model

```
imp_var <- c('temp_1','average','actual')

train_imp <- train_set[ , imp_var]
test_imp <- test_set[ , imp_var]
set.seed(22,sample.kind = "Rounding")
```

```
## Warning in set.seed(22, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
temp.rf_imp <- randomForest(actual ~ . , data = train_imp, mtry=2 ,ntree = 100,
                                          importance = TRUE, na.action = na.omit)
temp.rf_imp
```

```
##
## Call:
##  randomForest(formula = actual ~ ., data = train_imp, mtry = 2,      ntree = 100, importance = TRUE,
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 23.7083
##                     % Var explained: 82.86
```

Lets check the predictions of this model

```
pred_imp = predict(temp.rf_imp, newdata = test_imp)

#now lets calculate the absolute error between the predicted values and the actual value
abs_erre_rf_imp <- abs(pred_imp - base_actual_values_test)

#take the mean of absolute error
abs_mean_rf_imp <- mean(abs_erre_rf_imp)
abs_mean_rf_imp
```

```
## [1] 4.122023
```

```r
#calculate MAPE- Mean absolute percentage error
mape_imp = 100 * (abs_erre_rf_imp / base_actual_values_test)


#accuracy
accuracy_imp = 100 - mean(mape_imp)
accuracy_imp
```

```
## [1] 93.34582
```

Adding these values to results table

```r
predicted_results <- bind_rows(predicted_results, data_frame(Date = "31-12-2019" , Method = "Random_fore

predicted_results %>% knitr::kable()
```

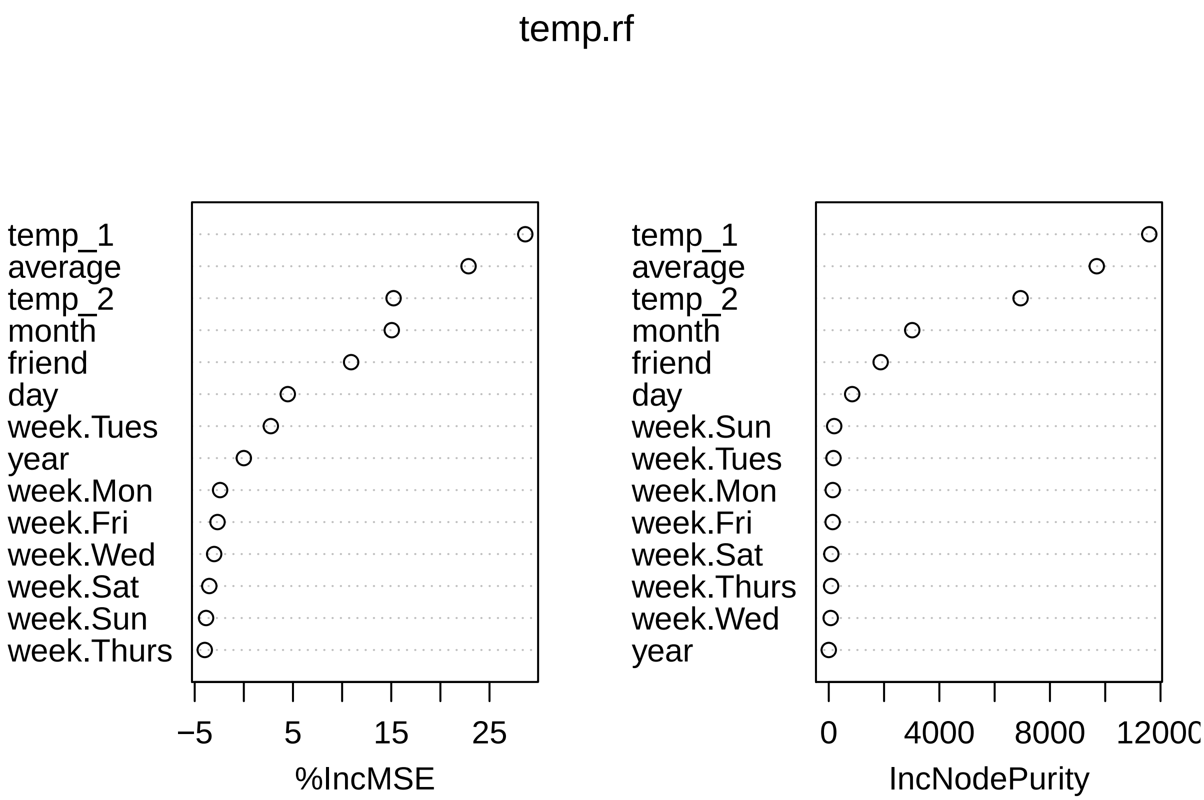
Date	Method	Error	Accuracy
31-12-2019	Average Model appraoch	4.515909	92.57712
31-12-2019	Random_forest	4.190482	93.26397
31-12-2019	Random_forest_improved	4.148032	93.31148
31-12-2019	Random_forest_with_imp_features	4.122023	93.34582

On taking just the imp variables/features we have seen that the performance is almost similar infact little improved as it is when all the variables were considered. In real life project, if we work on all the variables, model will take more time and memory and will give same result, hence it is advisable that before applying the model, we should identify the most important variables/features which will majorly contribute in the prediction.

7.3 Visualization -final after model

Lets visualize the importance of the features

```r
varImpPlot(temp.rf, sort = TRUE)
```

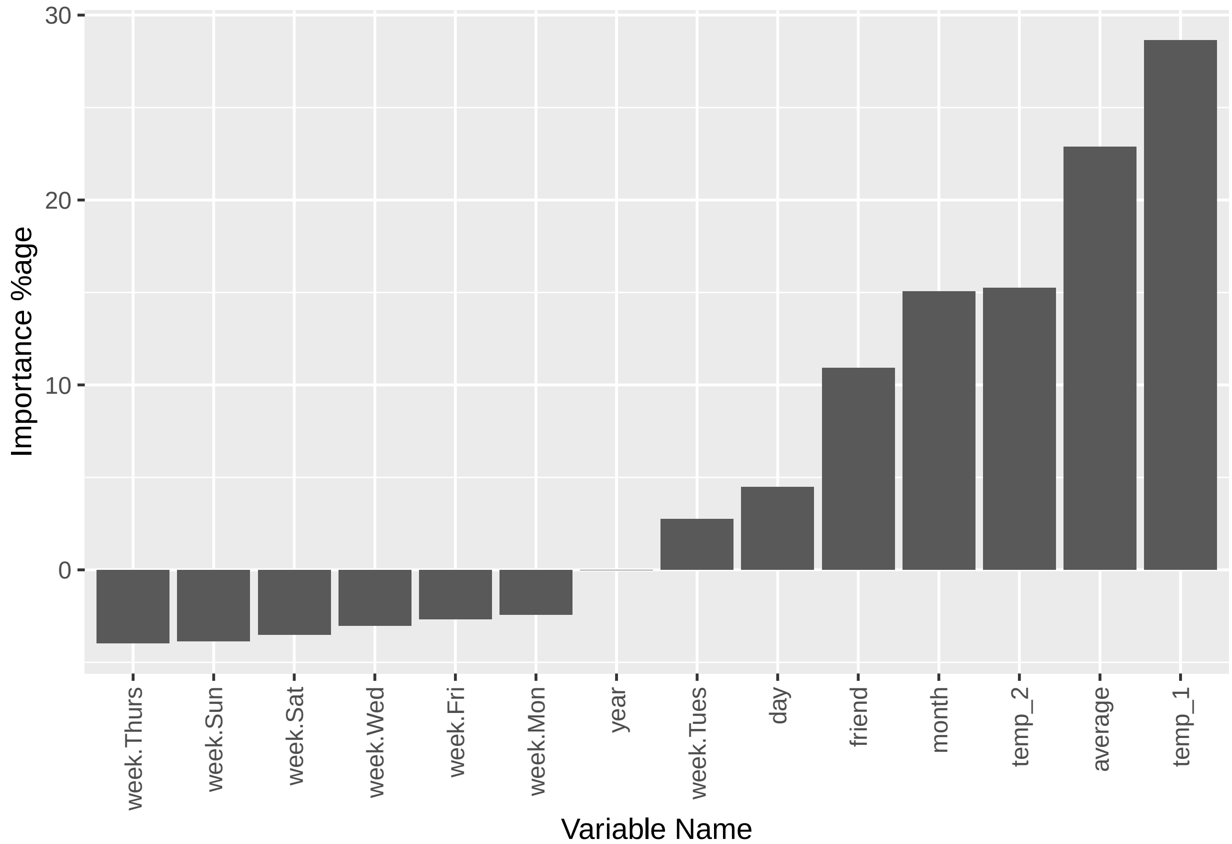


Lets visualize the same in historgram

```
imp1 = as.data.frame(importance(temp.rf))
imp1 = cbind(vars=rownames(imp1), imp1)
testing <- imp1[2]
imp1 = imp1[order(testing),]
imp1$vars = factor(imp1$vars, levels=unique(imp1$vars))
imp1 %>% ggplot(aes(imp1$vars ,imp1$`%IncMSE` )) + geom_bar(stat = "identity") + theme(axis.text.x = el
```

```
## Warning: Use of `imp1$vars` is discouraged. Use `vars` instead.
```

```
## Warning: Use of `imp1$`%IncMSE`` is discouraged. Use `%IncMSE` instead.
```



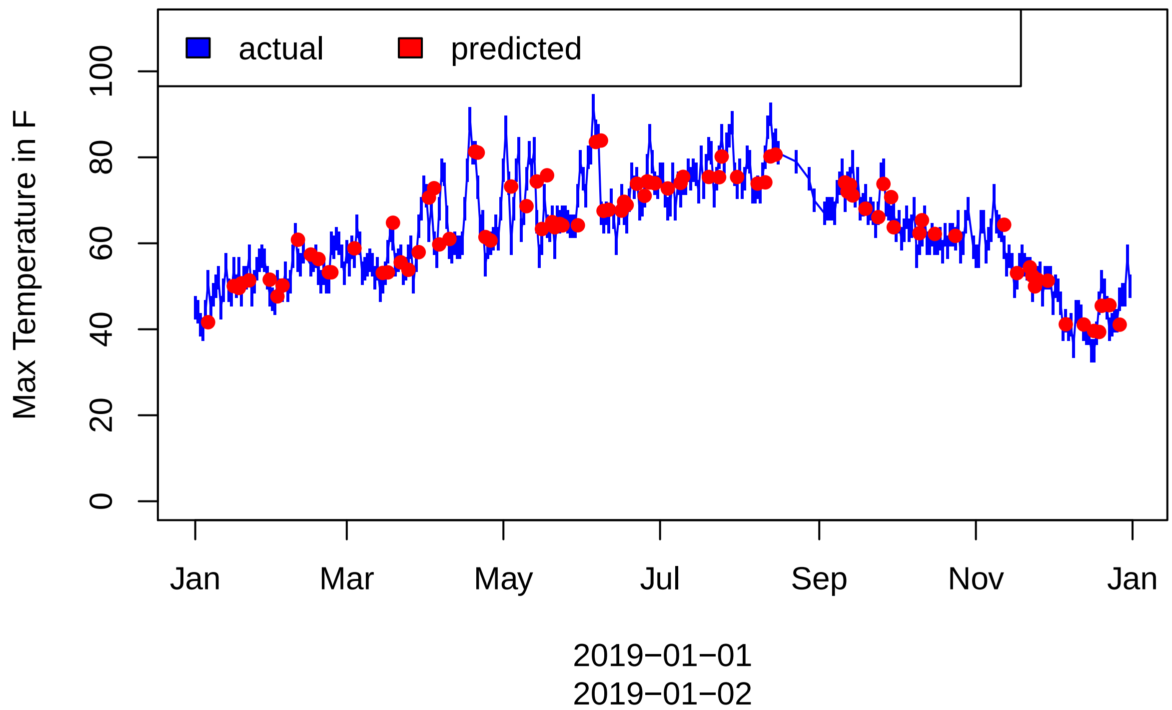
Lets plot the predictions vs actual and find out any outliers if present

```
actual_data <- date %>% select(date,actual)

#fetch data from test_set
years_t <- test_set$year
months_t <- test_set$month
day_t <- test_set$day

test_set_date <- test_set %>%
  mutate(date = make_date(year, month, day))
#make df
predicted_data <- test_set_date %>% select(date) %>% mutate(predictions = pred_imp)

plot(actual_data,date$date, type="o", col="blue", pch="l", lty=1, ylim=c(0,110), ylab="Max Temperature 
points(predicted_data,y=NULL,col="red", pch=16 )
legend(x="topleft",
       ncol = 4,
       legend = c("actual",
                  "predicted"
                  ),
       fill = c("blue","red")
  )
```



As seen, no outliers are present.

8 Results

The following table represents average model and different variants of Random Forest Model.

```
predicted_results %>% knitr::kable()
```

Date	Method	Error	Accuracy
31-12-2019	Average Model appraoch	4.515909	92.57712
31-12-2019	Random_forest	4.190482	93.26397
31-12-2019	Random_forest_improved	4.148032	93.31148
31-12-2019	Random_forest_with_imp_features	4.122023	93.34582

we therefore obsetved the best error and accuracy with the last model where only important features were considered to predict the max temperature. As per our model, the predicted temperature is 48.57 whereas the actual max temperature 50, which represents our model to be good.

9 Conclusion

So here in this project we used random forest to build a machine learning model for forecasting the max temperature of the last day of the year after using the historical 1 year data of temperature.

We observed that by tunning few hyperparameters we improved the model performance.

We also observed that the model preformance was almost the same infact little improved and the error value was also similar when we build the model using all the features and when we build the model by only using the two most important features. This can conclude that some features do not contribute much in preparing the prediction model and we can ignore those features for saving time.