A Jagdish Kumar

Bike rental count project

2nd September 2018

Contents

Chapter 1

Introduction

Project Name - Bike Renting

1.1 Data Set -

1) day.csv

1.2 Problem statement -

The objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings.

The details of data attributes in the dataset are as follows -
instant: Record index
dteday: Date
season: Season (1:springer, 2:summer, 3:fall, 4:winter)
yr: Year (0: 2011, 1:2012)
mnth: Month (1 to 12)
hr: Hour (0 to 23)
holiday: weather day is holiday or not (extracted fromHoliday Schedule)
weekday: Day of the week
workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
weathersit: (extracted fromFreemeteo)
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: Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min),
t_min=-8, t_max=+39 (only in hourly scale)
atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_maxt_
min), t_min=-16, t_max=+50 (only in hourly scale)
hum: Normalized humidity. The values are divided to 100 (max)
windspeed: Normalized wind speed. The values are divided to 67 (max)
casual: count of casual users
registered: count of registered users
cnt: count of total rental bikes including both casual and registered

Utilizing machine learning to forecast the demand for the bikes and using different types of regression to find an algorithm to predict the demand for bikes based on calenderic and weather information.

This project tries to create a forecasting function based on two years of historical data by utilizing the machine learning libraries scikit-learn

1.3 Problem Description

The goal is to forecast the demand for bikes in dependency of weather conditions like outside temperature and calendared information e.g. holidays. This information and the demand structure is provided in a set with two years of daily historic data.

The demand is given as the total daily demand and as a split for registered users and casual users. To increase the quality of the prediction registered user demand and casual user demand will be predicted separately in step two.

To make predictions machine learning is used to train regressors. I used linear regression for this kind of problem. In addition a decision tree regression, random forest regression, knn regression and **xgboost regression** have used for comparison.

Metrics

To measure the performance of the regressions four standard regression metrics are used: Mean squared error (MSE), Root Mean Squared Error (RMSE), the coefficient of determination (R^2) and Mean Absolute Percentage Error. All metrics are calculated for both regressor types. For comparison and parameter tuning only R^2 is used. "The RMSE is directly interpretable in terms of measurement units, and so is a better measure of goodness of fit than a correlation coefficient.

Chapter 2

Methodology

2.1 Pre Processing

2.1.1 Missing Values

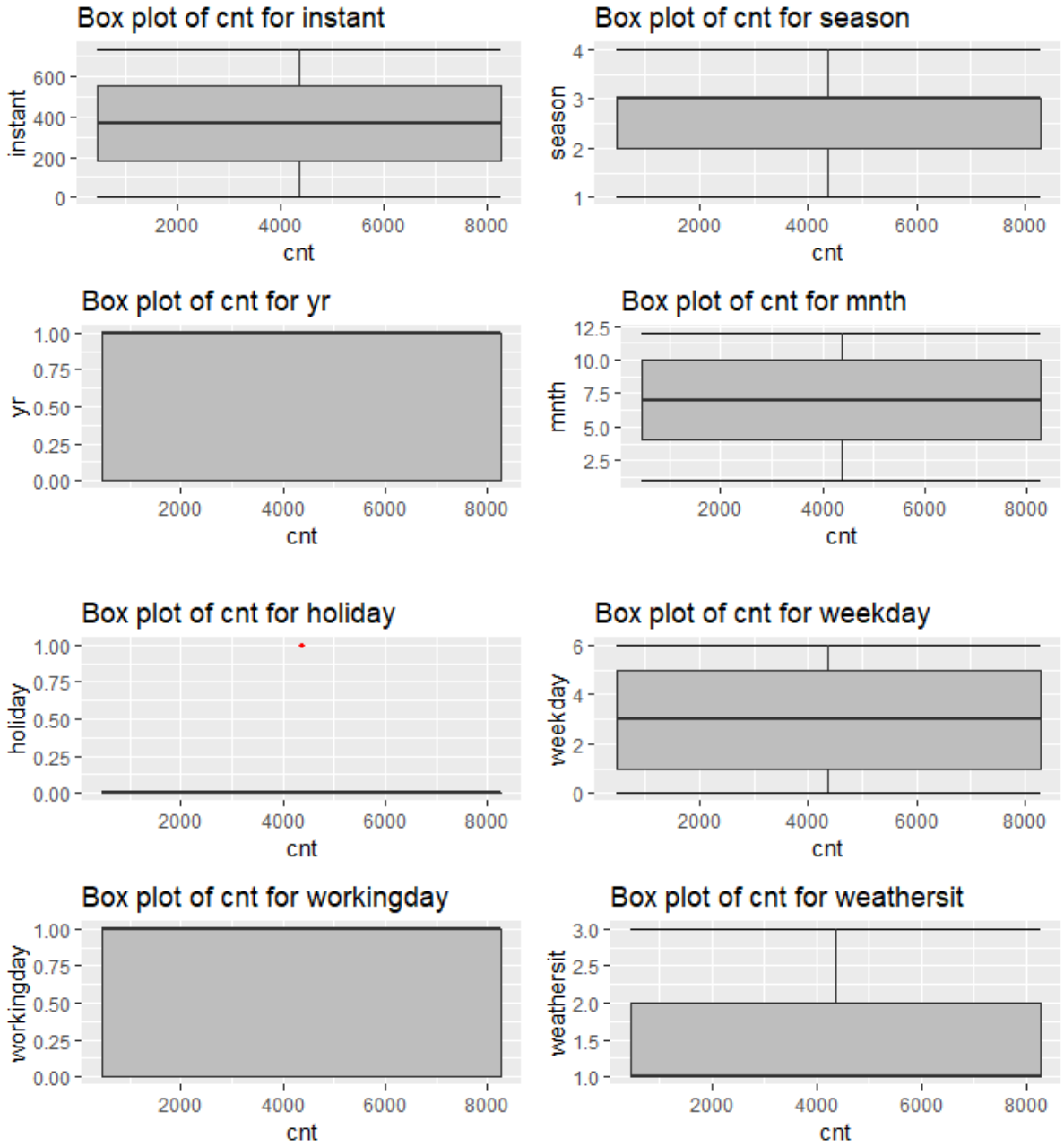
This dataset has no missing values which can be found with this line of code in R. The dataset is very concise.

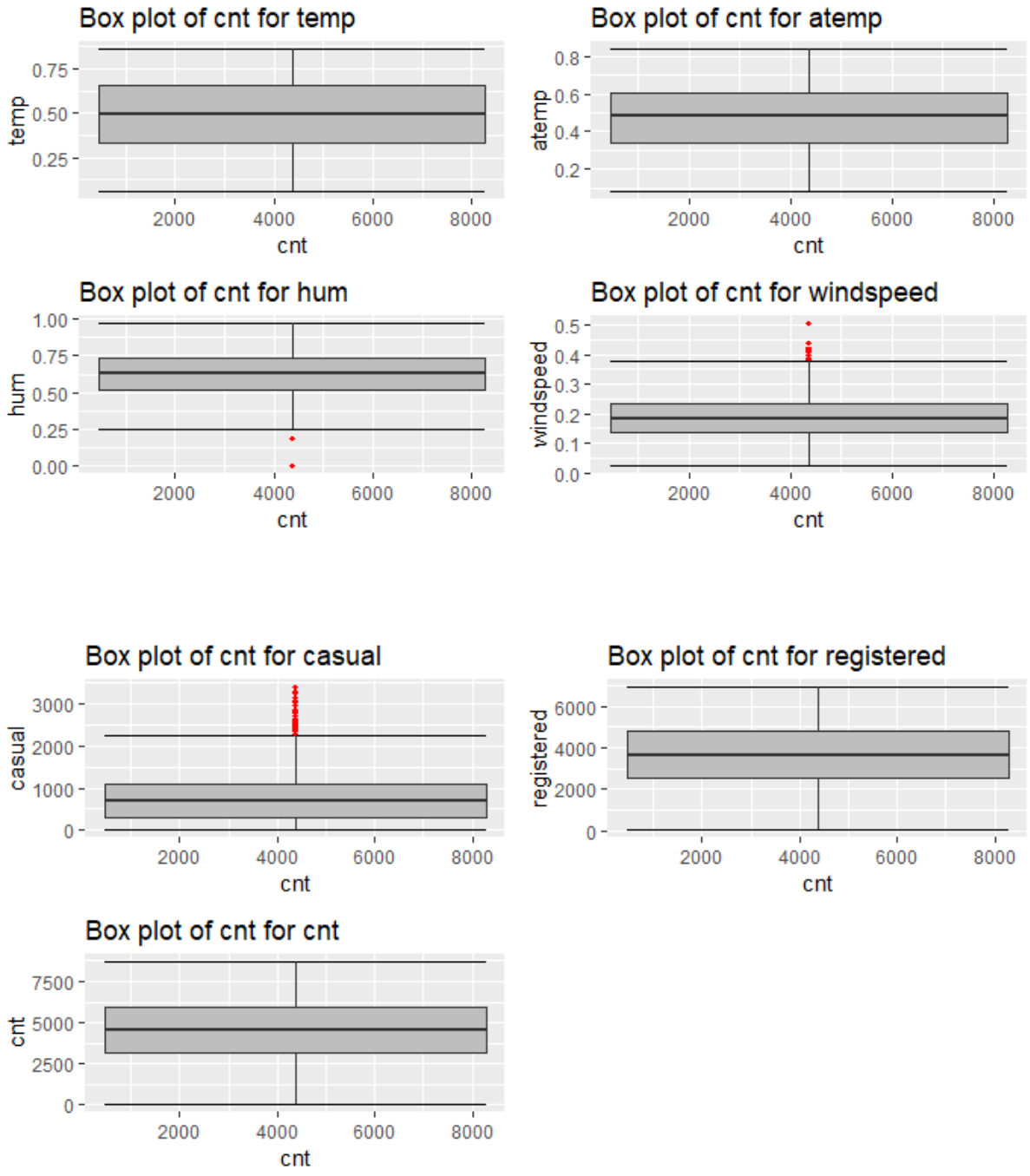
```
> missing_val
           apply.data..2..function.x...
instant                               0
dteday                                0
season                                0
yr                                    0
mnth                                  0
holiday                               0
weekday                               0
workingday                            0
weathersit                            0
temp                                  0
atemp                                 0
hum                                   0
windspeed                             0
casual                                0
registered                            0
cnt                                   0
```

Exploratory Visualization

2.1.2 Outlier Analysis

One of the other steps of **pre-processing** apart from checking for normality is the presence of outliers. In this case we use a classic approach of removing outliers, *Tukey's method*. We visualize the outliers using *boxplots*



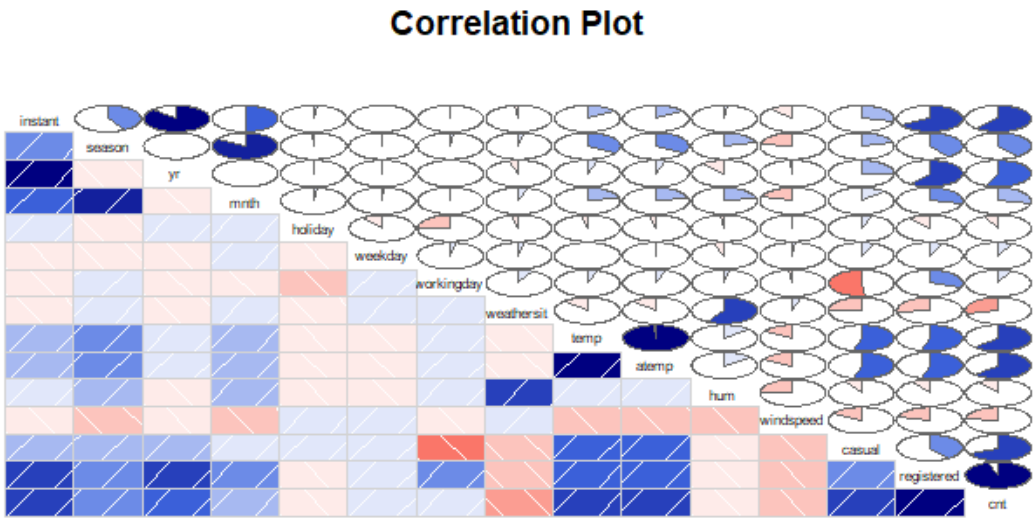


The visualization shows a classic seasonal pattern with an uptrend year over year. There are some outliers. These are left in the dataset because they are not due to measurement errors, but to extreme weather conditions; because extreme weather conditions are part of the problem, so the data is not excluded.

2.1.3 Feature Selection

Before performing any type of modelling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. We have used **heatmap** in python for feature selection and **corrplot** in R.

Based on the p-values, column's significance can be understood and hence those columns are dropped whose p-value was less than 0.05.



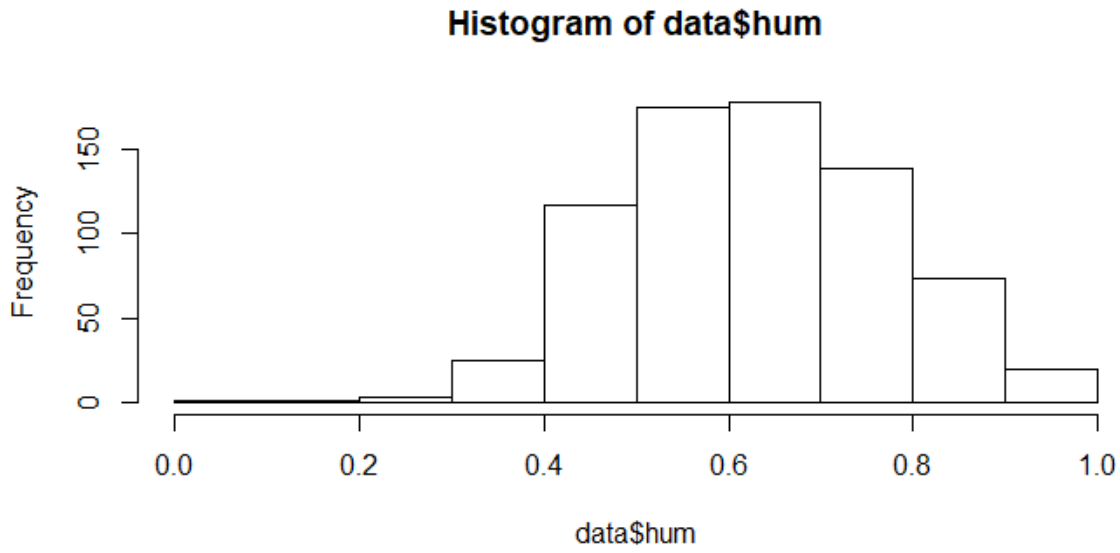
Multicollinearity check

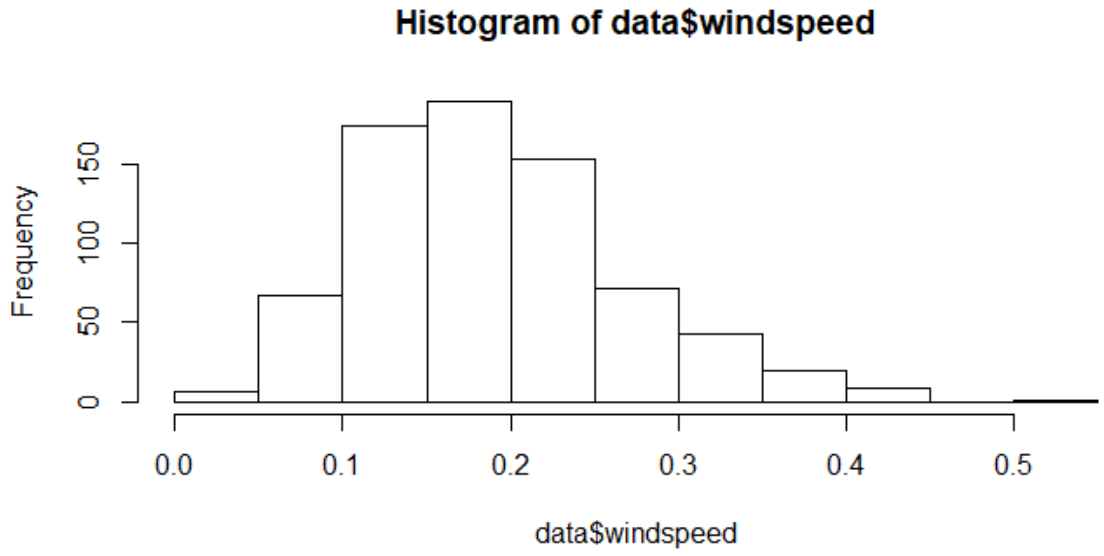
```
> vifcor(numeric_data[, -15], th = 0.9)
1 variables from the 14 input variables have collinearity problem:

atemp

After excluding the collinear variables, the linear correlation
coefficients ranges between:
min correlation ( weekday ~ instant ):  -1.617914e-05
max correlation ( yr ~ instant ):  0.8660254

---------- VIFs of the remained variables --------
    Variables        VIF
1     instant 583.414496
2      season   4.105162
3          yr 446.434580
4        mnth 148.339040
5     holiday   1.097188
6     weekday   1.047542
7  workingday   3.113128
```

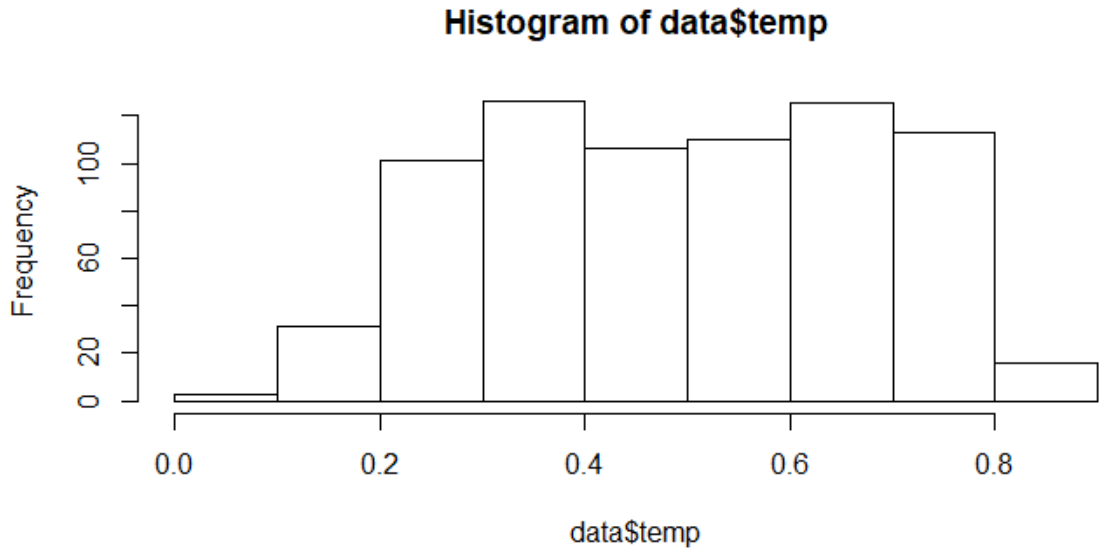
```
8  weathersit   1.929555
9        temp   2.478148
10        hum   1.941898
11  windspeed   1.225988
12     casual   3.531317
13 registered   6.002123
```

Categorical data like 'weekday' or 'working day' are already processed.

2.1.4 Normality Check







Most of the data is already normalized or binary.

Data Preprocessing (Methodology)

Dates get dropped because the regressor cannot read this datatype and the information is already stored in the 'mnth' and 'yr' feature.

2.2 Modelling

2.2.1 Model Selection

Model selection is the process of choosing between different machine learning approaches - e.g. Decision tree regression from rpart, linear regression, random forest, **xgboost**, etc - or choosing between different hyperparameters or sets of features for the same machine learning approach - e.g. deciding between the polynomial degrees/complexities for linear regression.

There may be certain qualities you look for in a model:

- ✓ Interpretable
- ✓ Simple - easy to explain and understand
- ✓ Accurate
- ✓ Fast (to train and test)
- ✓ Scalable

2.2.2 Decision tree

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**.

The core algorithm for building decision trees called **ID3** by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with *Standard Deviation Reduction*.

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). We use standard deviation to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous its standard deviation is zero.

The standard deviation reduction is based on the decrease in standard deviation after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest standard deviation reduction (i.e., the most homogeneous branches)

2.2.3 Linear Regression

Simple linear regression is useful for finding relationship between two continuous variables. One is predictor or independent variable and other is response or dependent variable. It looks for statistical relationship but not deterministic relationship. Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other. For example, using temperature in degree Celsius it is possible to accurately predict Fahrenheit. Statistical relationship is not accurate in determining relationship between two variables. For example, relationship between height and weight.

The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible.

$$Y(pred) = b0 + b1*x$$

The values b0 and b1 must be chosen so that they minimize the error. If sum of squared error is taken as a metric to evaluate the model, then goal to obtain a line that best reduces the error.

$$\text{Error} = \sum_{i=1}^{n} (actual\_output - predicted\_output) ** 2$$

If we don't square the error, then positive and negative point will cancel out each other.

For model with one predictor, Intercept Calculation is done using,

$$b_0 = \bar{y} - b_1\bar{x}$$

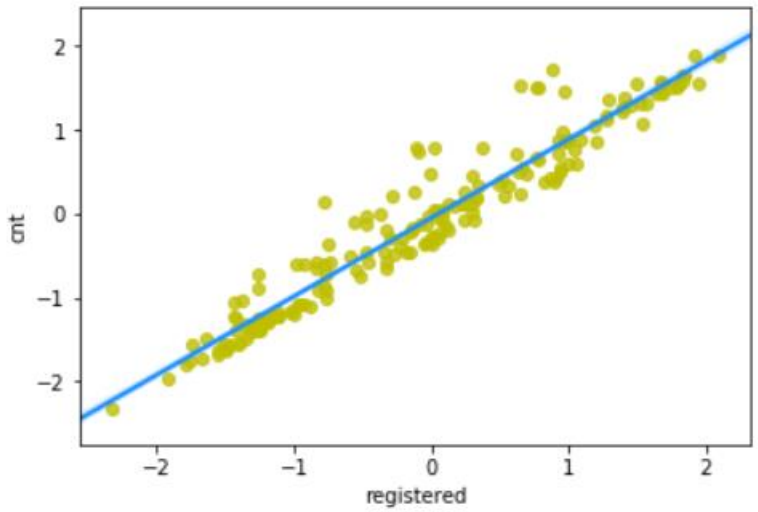


Fig : Linear regression performed on registered and cnt(response) column.

2.2.4 Random Forest

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyperparameter tuning, a great result most of the time. It is also one of the most used algorithms, because its simplicity and the fact that it can be used for both classification and regression tasks.

Random Forest is a supervised learning algorithm. Like you can already see from it's name, it creates a forest and makes it somehow random. The „forest" it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

2.2.5 XGBoost Regressor

XGBoost stands for eXtreme Gradient Boosting.XGBoost is an algorithm that has recently been dominating applied machine learning.

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines.

Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made.

Chapter 3

Conclusion

3.1 Model Evaluation

Using sklearn library in python and MLmetrics in R, we have calculated the R-square, Mean absolute error, Root mean square error, and mean absolute percentage error of our model. Higher the value of r square better the model and lower the value of MAE, RMSE and MAPE better the model. MAE, MAPE and RMSE denotes error, So their small values are desirable.

```
> models_df
         model_name rsq_value rmse_value mae_value  mape_value
1     Decision Tree 0.7124185 1050.77624 813.62627 0.273103667
2 linear regression 0.7943580  885.76505 671.87916 0.198351958
3     Random Forest 0.8893730  651.96356 466.66609 0.149788997
4 xgboost regressor 0.9993922   48.09633  34.21515 0.009591292
```

Appendix

R code

```
setwd("D:/edwisor/Projects/bike rental")
data <- read.csv("day.csv", header = T)

#Loading required libraries
library(corrgram)
library(MLmetrics)
library(Metrics)
library(rsq)
library(caret)
library(rpart)
library(usdm)
library(randomForest)
library(xgboost)


#checking for missing values
missing_val = data.frame(apply(data,2,function(x){sum(is.na(x))}))
missing_val

# ## BoxPlots - Distribution and Outlier Check
numeric_index = sapply(data,is.numeric) #selecting only numeric

numeric_data = data[,numeric_index]

cnames = colnames(numeric_data)

for (i in 1:length(cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "cnt"), data = subset(data))+
           stat_boxplot(geom = "errorbar", width = 0.5) +
           geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                        outlier.size=1, notch=FALSE) +
           theme(legend.position="bottom")+
           labs(y=cnames[i],x="cnt")+
           ggtitle(paste("Box plot of cnt for",cnames[i])))
}

# ## Plotting plots together
gridExtra::grid.arrange(gn1, gn2, gn3, gn4, nrow=2, ncol=2)
gridExtra::grid.arrange(gn5, gn6, gn7, gn8, nrow=2, ncol=2)
gridExtra::grid.arrange(gn9, gn10, gn11, gn12, nrow=2, ncol=2)
gridExtra::grid.arrange(gn13, gn14, gn15, nrow =2, ncol=2)
```

```
#correlation plot
corrgram(data[,numeric_index], order = F,
       upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")


#check multicolinearity
vif(numeric_data[, -15])
vifcor(numeric_data[, -15], th = 0.9)

## Dimension Reduction
data = subset(data, select = -c(dteday, instant, atemp, casual, registered))

#Normality check
hist(data$hum)
hist(data$windspeed)
hist(data$temp)

#Divide the data into train and test
#set.seed(123)
train_index = sample(1:nrow(data), 0.8 * nrow(data))
train = data[train_index,]
test = data[-train_index,]

###model building
model_name = vector()
rsq_value = vector()
rmse_value = vector()
mae_value = vector()

register_model_score = function(name, rsq, rmse, mae){
  model_name <<- append(model_name, name)
  rsq_value <<- append(rsq_value, rsq)
  rmse_value <<- append(rmse_value, rmse)
  mae_value <<- append(mae_value, mae)

}

print_score <- function(postResample_score, model_name, register_model = 1){
  rsq = postResample_score[[2]]
  mae = postResample_score[[3]]
  rmse_val = postResample_score[[1]]

  cat(c("R-sq=", rsq, "\n"))
  cat(c("RMSE =", rmse_val, "\n"))
  cat(c("MAE =", mae, "\n"))

  if(register_model ==1){
    register_model_score(model_name, rsq, rmse_val, mae)
  }
```

```
}

#calculate MAPE
mape_value = vector()
Mape = function(y, yhat){
  mean(abs((y - yhat)/y))
  mape_value <<- append(mape_value, mape)
  cat(c("MAPE =", mape, "\n"))
}


# ##rpart for regression
name1 = "Decision Tree"
fit = rpart(cnt ~ ., data = train, method = "anova")
predictions_DT = predict(fit, test[,-11])
score1 = postResample(predictions_DT, test$cnt)
print_score(score1, name1)
mape_DT <- mape(test$cnt, predictions_DT)
mape_value <<- append(mape_value, mape_DT)

#linear regression
name2 = "linear regression"
LR_model = lm(cnt ~., data = train)
summary(LR_model)
predictions_LR = predict(LR_model, test[, 1:10])
score2 = postResample(predictions_LR, test$cnt)
print_score(score2, name2)
mape_LR <- mape(test$cnt, predictions_LR)
mape_value <<- append(mape_value, mape_LR)

#KNN regressor
#name3 = "KNN Regression"
#y = train$cnt
#Knn_pred = FNN::knn.reg(train, test, y, k=1, algorithm=c("kd_tree", "cover_tree", "brute"))
#score3 = postResample(Knn_pred, test$cnt)
#print_score(score3, name3)
#mape_knn <- mape(test$cnt, knn_pred)

#random forest
name4 = "Random Forest"
RF = randomForest(cnt ~ ., data = train, mtry = 4, importance = TRUE, ntress = 500)
predictions_RF = predict(RF, test[,-11])
score4 = postResample(predictions_RF, test$cnt)
print_score(score4, name4)
mape_RF <- mape(test$cnt, predictions_RF)
mape_value <<- append(mape_value, mape_RF)

#xgboost
name5 = "xgboost regressor"
bike_model_xgb <- xgboost(data = as.matrix(train), # training data as matrix
                          label = train$cnt,  # column of outcomes
```

```
                nrounds = 84,      # number of trees to build
                objective = "reg:linear", # objective
                eta = 0.3,
                depth = 6,
                verbose = 0  # silent
)

# Make predictions
xgb_pred <- predict(bike_model_xgb, as.matrix(test))
score5 = postResample(xgb_pred, test$cnt)
print_score(score5, name5)
mape_xgb <- mape(test$cnt, xgb_pred)
mape_value <<- append(mape_value, mape_xgb)


###Model summary
models_df = data.frame(model_name, rsq_value, rmse_value, mae_value, mape_value)
models_df
```

Python code

In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import calendar
import seaborn as sns
import scipy as sp

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

In [2]:
```
os.chdir("D:/edwisor/Projects/bike rental")
```

In [3]:
```
data = pd.read_csv("day.csv")
```

In [4]:
```
data.describe()
```

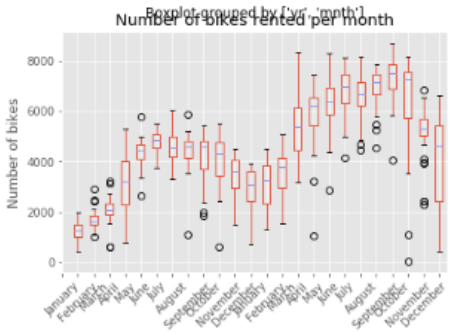
Out[4]:

	instant	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.(
mean	366.000000	2.496580	0.500684	6.519836	0.028728	2.997264	0.683995	1.395349	0.495385	0.474354	0.627894	0.190486	848.(
std	211.165812	1.110807	0.500342	3.451913	0.167155	2.004787	0.465233	0.544894	0.183051	0.162961	0.142429	0.077498	686.(
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.059130	0.079070	0.000000	0.022392	2.(
25%	183.500000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000	0.337083	0.337842	0.520000	0.134950	315.(
50%	366.000000	3.000000	1.000000	7.000000	0.000000	3.000000	1.000000	1.000000	0.498333	0.486733	0.626667	0.180975	713.(
75%	548.500000	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	2.000000	0.655417	0.608602	0.730209	0.233214	1096.(
max	731.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000	0.861667	0.840896	0.972500	0.507463	3410.(

Characteristics:

The dataset is very concise and missing values are not a problem. Most of the data is already normalized or binary. Categorical data like 'weekday' or 'working day' are already processed.

In [5]:
```
# Visulazation

plt.style.use('ggplot')

data.boxplot(column='cnt', by=['yr','mnth'])

plt.title('Number of bikes rented per month')
plt.xlabel('')
plt.xticks((np.arange(0,len(data)/30,len(data)/731)), calendar.month_name[1:13]*2, rotation=45)
plt.ylabel('Number of bikes')

plt.show()
```

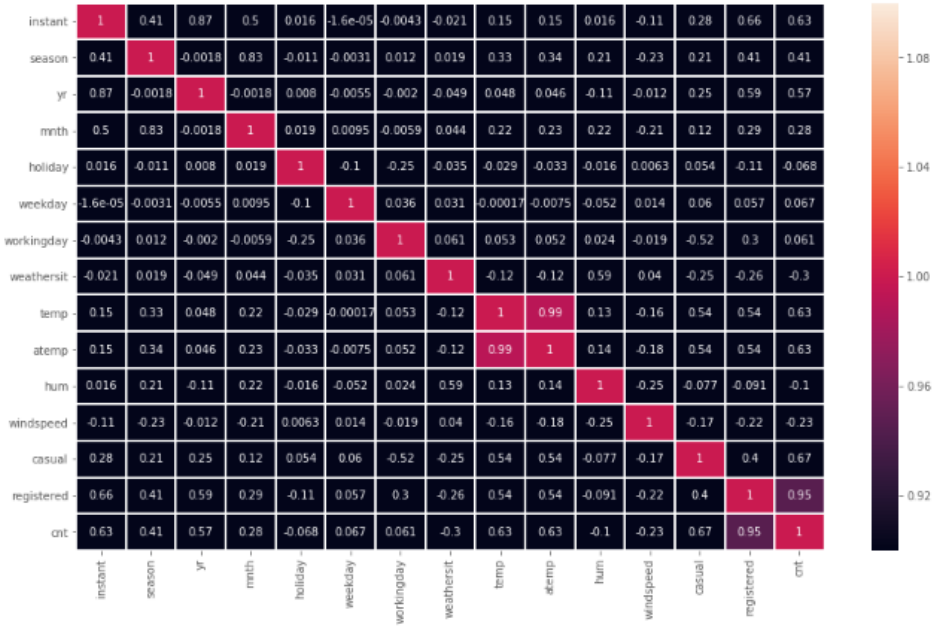


Exploratory Visualization

The visualization shows a classic seasonal pattern with an up trend year over year. There are some outliers. These are left in the dataset because they are not due to measurement errors, but to extreme weather conditions. Because extreme weather conditions are part of the problem, so the data is not excluded.

In [6]:
```
#Feature Selection
#heat map
plt.figure(figsize=(15, 9))
corr = data.corr()
sns.heatmap(corr, annot=True, linewidths=1, vmin=1, vmax=1)
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x22876b4aa58>



Data Preprocessing (Methodology)

Dates get dropped because the regressor can not read this datatype and the information is already stored in the 'mnth' and 'yr' feature. As 'temp' and 'atemp' are highly correlated from heatmap, "atemp" is dropped.

In [7]:
```
data = data.drop(columns = ["dteday", "instant", "atemp", "casual", "registered"])
```

In [8]:
```
train, test = train_test_split(data, test_size =0.2)
```

In [9]:
```
#MODEL BUILDING
Rsq_value = []
MAE_value = []
RMSE_value = []
MAPE_value = []

def register_model_score(rsq, mae, rmse, mape):
    Rsq_value.append(rsq)
    MAE_value.append(mae)
    RMSE_value.append(rmse)
    MAPE_value.append(mape)

def generate_score(y_test, predictions):
    rsq = r2_score(y_true = y_test, y_pred = predictions)
    mse = mean_squared_error(y_true = y_test, y_pred = predictions)
    mae = mean_absolute_error(y_test, predictions)
    rmse = np.sqrt(mse)
    mape = np.mean(np.abs((y_test - predictions)/y_test))
    print("R sq ="+ str(rsq))
    print("MAE ="+ str(mae))
    print("RMSE = "+ str(rmse))
    print("MAPE =" + str(mape))
    return(rsq, mae, rmse, mape)
```

In [10]:
```
#Decision Tree
fit_DT = DecisionTreeRegressor(max_depth=2).fit(train.iloc[:,0:10], train.iloc[:,10])

#predict new test cases
predictions_DT = fit_DT.predict(test.iloc[:,0:10])
```

In [11]:
```
#model score
rsq_val, mse_val, rmse_val, mape_val = generate_score(test.iloc[:,10], predictions_DT)

register_model_score( rsq_val, mse_val, rmse_val, mape_val)
```

```
R sq =0.6807864583327976
MAE =807.8507321372605
RMSE = 1018.8405158221319
MAPE =0.25040351790922394
```

In [12]:
```
#Linear regression
LR = LinearRegression()
LR.fit(train.iloc[:,0:10], train.iloc[:,10])
```

Out[12]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [13]:
```
LR_Pred = LR.predict(test.iloc[:,0:10])
```

In [14]:
```
#model score
rsq_val, mse_val, rmse_val, mape_val = generate_score(test.iloc[:,10], LR_Pred)
register_model_score( rsq_val, mse_val, rmse_val, mape_val)
```

```
R sq =0.782082841095297
MAE =646.2760573292974
RMSE = 841.8045480119823
MAPE =0.18485528710701032
```

In [15]:
```
#Random forest
RF_model = RandomForestRegressor(n_estimators = 500).fit(train.iloc[:,0:10], train.iloc[:,10])
RF_predictions = RF_model.predict(test.iloc[:,0:10])
```

In [16]:
```
#model score
rsq_val, mse_val, rmse_val, mape_val = generate_score(test.iloc[:,10], RF_predictions)
register_model_score(rsq_val, mse_val, rmse_val, mape_val)
```

```
R sq =0.8769787558725319
MAE =438.50473469387754
RMSE = 632.492836435183
MAPE =0.12947343705710504
```

In [17]:
```
#KNN
knn_model = KNeighborsRegressor(n_neighbors =3).fit(train.iloc[:,0:10], train.iloc[:,10])
knn_predictions = knn_model.predict(test.iloc[:,0:10])
```

In [18]:
```
#model score
rsq_val, mse_val, rmse_val, mape_val = generate_score(test.iloc[:,10], knn_predictions)
register_model_score(rsq_val, mse_val, rmse_val, mape_val)
```

```
R sq =0.7072912682503427
MAE =702.2789115646259
RMSE = 975.6260609907639
MAPE =0.22221237210292238
```

In [19]:
```
#XGBoost regressor
reg = XGBRegressor(n_estimators = 500, seed =124, learning_rate = 0.1)
reg.fit(train.iloc[:,0:10], train.iloc[:,10])
```

Out[19]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=500,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=124,
       silent=True, subsample=1)
```

In [20]:
```
XGB_predictions = reg.predict(test.iloc[:,0:10])
```

In [21]:
```
#model score
rsq_val, mse_val, rmse_val, mape_val = generate_score(test.iloc[:,10], XGB_predictions)
register_model_score(rsq_val, mse_val, rmse_val, mape_val)
```

```
R sq =0.8615736239903539
MAE =458.1342623963648
RMSE = 670.9265502312273
MAPE =0.1310682897382689
```

In [22]:
```
Model_name = ["Decision Tree", "Linear Regression", "Random Forest", "KNN Regressor", "XGB Regressor"]
```

In [23]:
```
#Model summary
df = pd.DataFrame({"Model_name" : Model_name,
                   "R-sq":  Rsq_value,
                   "MAE":  MAE_value,
                   "RMSE": RMSE_value,
                   "MAPE": MAPE_value},
                      columns = ['Model_name', 'R-sq', 'MAE', 'RMSE', 'MAPE'])
df
```

Out[23]:

	Model_name	R-sq	MAE	RMSE	MAPE
0	Decision Tree	0.680786	807.850732	1018.840516	0.250404
1	Linear Regression	0.782083	646.276057	841.804548	0.184855
2	Random Forest	0.876979	438.504735	632.492836	0.129473
3	KNN Regressor	0.707291	702.278912	975.626061	0.222212
4	XGB Regressor	0.861574	458.134262	670.926550	0.131068