**1. INTRODUCTION**

**1.1 Problem statement**

A bike rental company is one which rents bikes for short periods of time. The objective of this Case is to Predict the 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

**1.2 Objective**

Our task is to build data models which will predict the count of bike rented

depending on various environmental and seasonal conditions. Hence **cnt** is our dependent variable. We have the data provided to us which has 737 observations and 16 variables in which **cnt** is a dependent variable and the remaining 15 variables are independent variables.

Given below is sample of the data set that we have been provided to predict the count of bike rents:

Table 1.1 Showing sample data with 9 columns

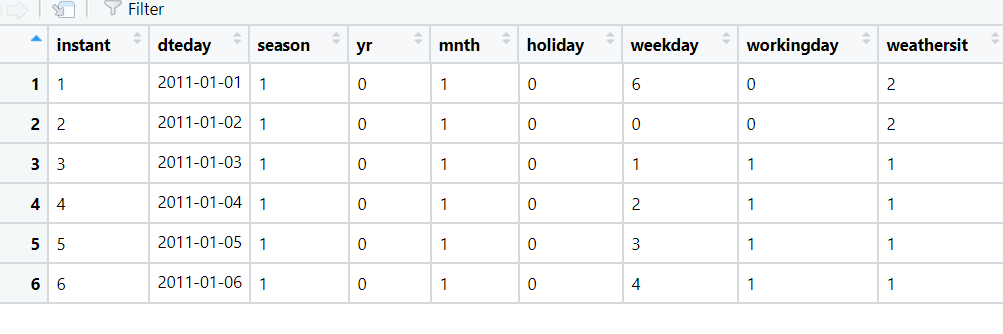


Table 1.2 Showing sample data with remaining 7 columns

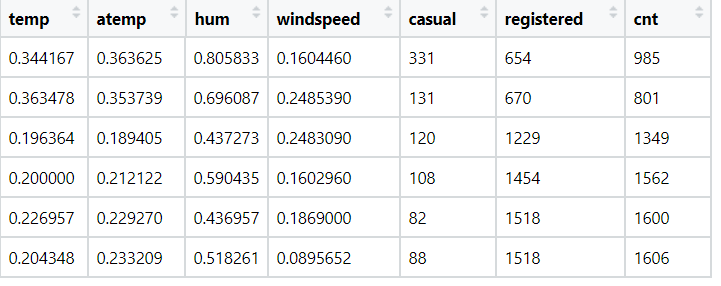
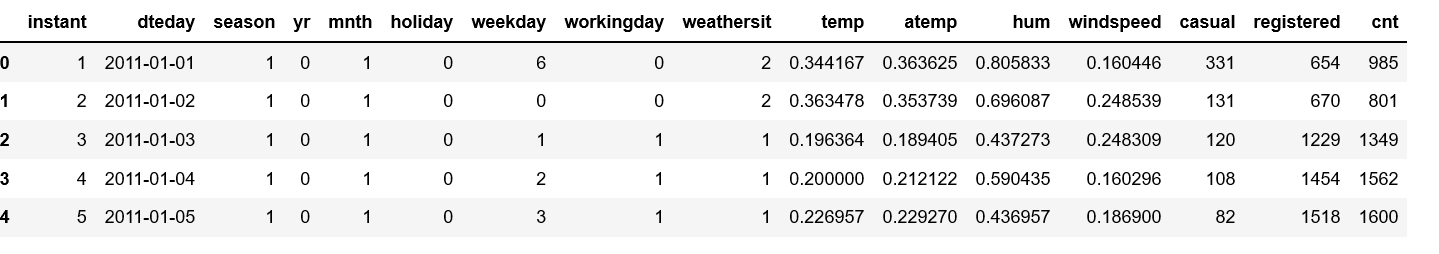


Table 1.3 Showing sample data in Panda



No of variables present in the data : 16

Independent variables : **instant, dteday, season, yr, mnth, holiday, weekday, workingday, weathersit, temp, atemp, hum, windspeed, casual, registered**

Dependent variable : **cnt**

**2. Data Pre-Processing**

**2.1 Exploratory Data Analysis**

Before creating a model to predict the bike rental count for the data provided it is important we clean the data and do the exploratory data analysis in order to have a clean data for the models being developed ahead.

1. At the verification of the data done from the previous step above, we find that the variable **instant** is just carrying an index value and it is not adding anything meaningful to the data so we can drop the variable.
2. Converted the following independent variables into categorical variables by converting the data type from int to factor:

**season, yr, mnth, holiday, weekday, workingday, weathersit**

1. The independent variable **dteday** is of type character having a string data with Month Date and Year. Since we already have the month and year in our data we are converting this character string into a date variable having only the date value and converting into a factor from 1 to 31 levels.
2. Dependent Variable **cnt** is a sum total of independent variable values of **casual** and **registered**. Hence we can drop the variables **casual** and **registered**

Post this initial analysis we have now 13 variables out of which we have 12 independent variables and 1 dependent variable.

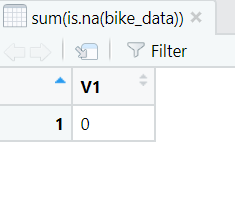
Independent variables : **dteday, season, yr, mnth, holiday, weekday, workingday, weathersit, temp, atemp, hum, windspeed**

Dependent variable : **cnt**

**2.2 Missing Value analysis**

**Missing Values**

First we calculate the no of NA values in the data set. We find that there are no missing values in the data. Hence we are not going to do the missing value analysis.



Python



**2.3 Outlier Analysis**

Plotting the boxplot for Outliers on independent Continuous variables**, temp, atemp, hum, windspeed,** we get the following results depicted below.

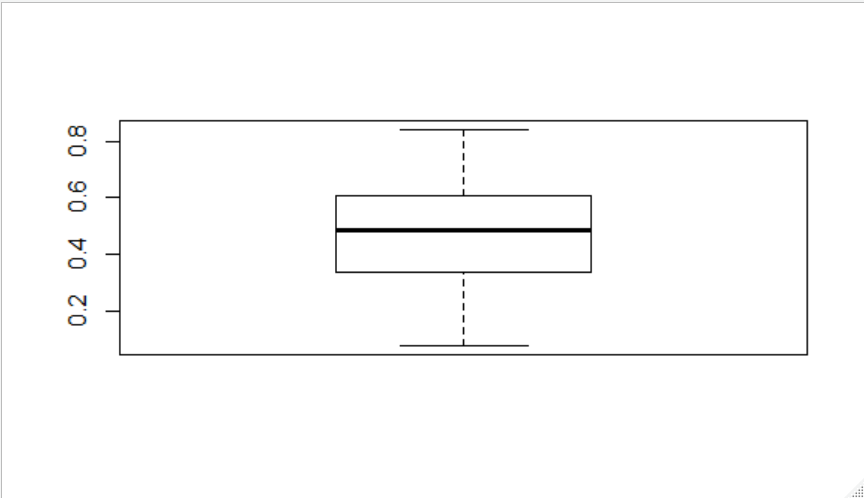


Fig 2.3.1 Boxplot for outlier analysis of variable **atemp**

**Python:**

A close up of a logo

Description automatically generated

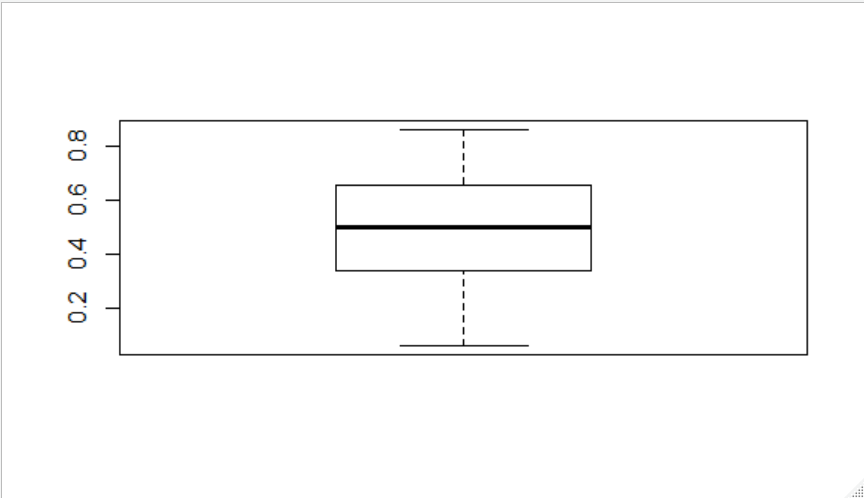


Fig 2.3.2 Boxplot for Outlier analysis of variable **temp**

**Python:**

**A close up of a logo

Description automatically generated**

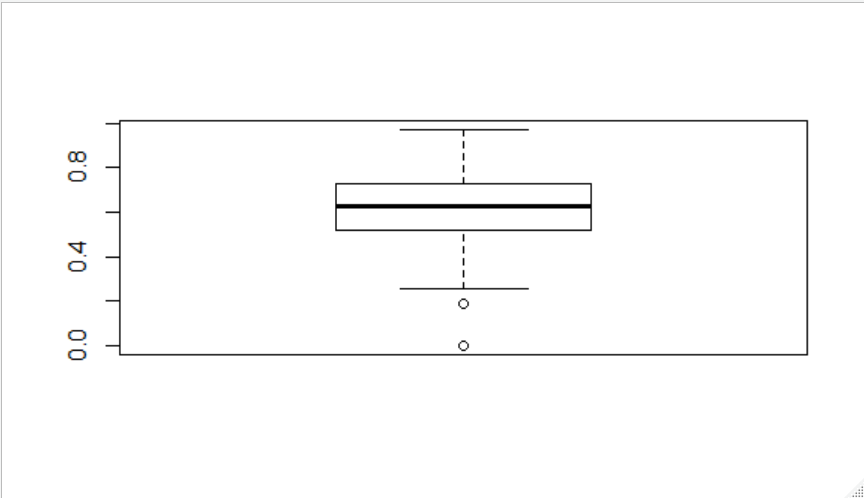


Fig 2.3.3 Boxplot for Outlier analysis of variable **hum**

**Python:**

**A close up of a logo

Description automatically generated**

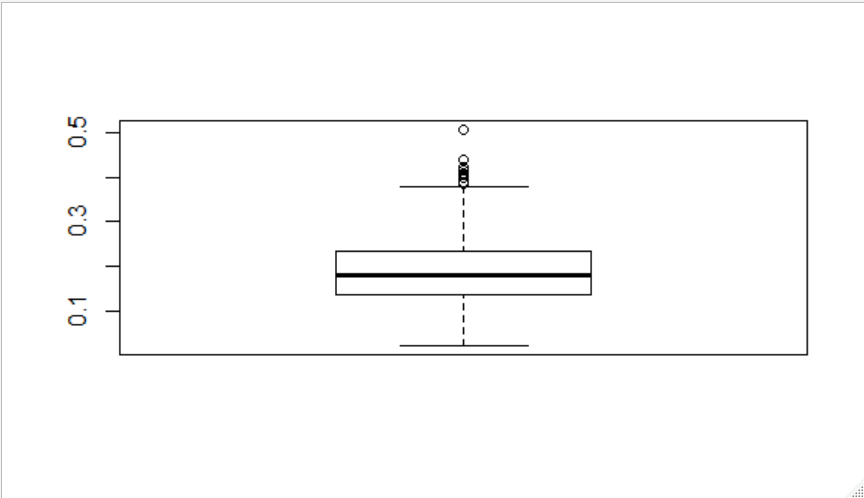


Fig 2.3.3 Boxplot for Outlier analysis of variable **windspeed**

**Python:**

**A close up of a logo

Description automatically generated**

Based on the plotted graph above for these variables we find that there are outliers for variables **hum** & **windspeed** only. There are no outliers for the **temp** and **atemp** variables. Hence we are going to perform outlier analysis on **hum** and **windspeed** using boxplot method and replacing them with NA and imputing these missing values of NA using KNN imputation method.

**2.4 Feature Selection**

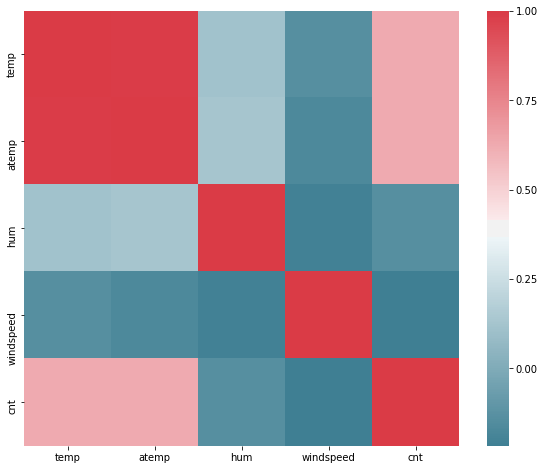
Feature selection is done to select subsets of relevant features (variables, predictors) to be used while designing the models. Here our dependent variable is continuous, so we are going for correlation check. Hence we plot the correlation plot between all the variables and below is the result we get.

A close up of a logo

Description automatically generated

Fig 2.4.1 Correlation plot for all the continuous variables.

**Python**:



From the above Correlation plot we can see that the variables **atemp** and **temp** are equally correlated with dependent variable **cnt** and are also highly correlated with each other. Hence we can remove one of them as both variables are contributing same data to the data set. So we will be removing the **atemp** variablefrom the data set.

We can verify the dependency of the continuous variables on the dependent variable **cnt** using scatter plot.

Fig 2.4.2 Scatter plot of **cnt** variable against **hum** variable

A screenshot of a cell phone

Description automatically generated

Fig 2.4.2 Scatter plot of **cnt** variable against **temp** variable

A screenshot of a cell phone

Description automatically generated

Fig 2.4.3 Scatter plot of **cnt** variable against **hum** variable

A screenshot of a cell phone

Description automatically generated

Fig 2.4.3 Scatter plot of **cnt** variable against **windspeed** variable

**Python:**

**A screenshot of a cell phone

Description automatically generated**

Fig 2.4.4 Humidity and feel temperature plot against bike rental count

A close up of a sign

Description automatically generated

Fig 2.4.4 Windspeed and temperature plot against bike rental count

From the above scatter plots we can verify that there is a positive relationship which is in linear nature between the bikes count and temperature. As temperature increases more people opt for bike ride hence there is a increase in bike count. Also there is a negative relationship between bike count against humidity and windspeed. As the humidity and windspeed increases the bike count is seen having lower counts.

**2.5 Feature Scaling**

The feature scaling is done for standardizing the independent features in the data set during data pre-processing. Since the values provided for continuous independent variables are already in normalized values we will be going ahead with the existing data itself.

**3. Model Evaluation**

Since our target variable to predict bike count , **cnt** is continous variable we will be evaluating the model using regression algorithms to predict the data which are as following,

Linear Regression

Decision Tree

Random Forest

First we split the data into train and test using sampling method and apply the regression models for the test data to predict the values and compare with the actual values.

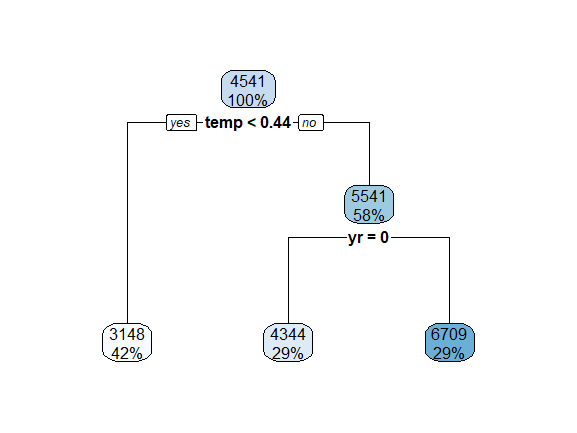
Here we are choosing RMSE and MAPE as the parameters for evaluating the model.

**Hyper Paratemeter tuning**

**Decision tree Model:**

Decision tree model are prone to overfitting hence we use the complexity parameter to use the constraint on the overfitting. Complexity parameter is the minimum improvement of the model needed at each node.

Initially we plot the Decision tree model against a cp value of cp=0.2 in rpart function.



**Fig 3.1.1 Decision tree split for a complex parameter(cp) value of 0.2**

Now we decrease the value of cp to a lesser value and try and plot the decision tree split.

A close up of a logo

Description automatically generated

**Fig 3.1.2 Decision tree split for a complex parameter(cp) value of 0.002**

Here we can see that the Decision tree model is overfitted due to the very low value of cp. Hence we need to identify the right value of complex parameter by doing Pruning.

Below is the plot for the cp for the Decision tree model.

A close up of a device

Description automatically generated

Fig 3.1.3 Plotting of the CV relative error against CP.

From the above diagram e see that the range of the CV relative error becomes stable around certain values and we can choose to select one of the value in this range for cp.

Also we can print the cp value for the decision tree model using printcp function and look at the errors.

CP nsplit rel error xerror xstd

1 0.3776748 0 1.000000 1.00411 0.045611

2 0.2207485 1 0.622325 0.65160 0.031916

3 0.0928550 2 0.401577 0.43282 0.031730

4 0.0489474 3 0.308722 0.33492 0.028107

5 0.0288305 4 0.259774 0.28796 0.027524

6 0.0159679 5 0.230944 0.25983 0.027592

7 0.0155408 6 0.214976 0.26201 0.024940

8 0.0146328 7 0.199435 0.25899 0.024917

9 0.0124511 8 0.184802 0.25262 0.024762

10 0.0102805 9 0.172351 0.24158 0.024418

11 0.0070989 10 0.162071 0.24425 0.024971

12 0.0068787 11 0.154972 0.25420 0.026378

13 0.0066160 12 0.148093 0.25157 0.026264

14 0.0051593 13 0.141477 0.25595 0.026561

15 0.0051292 14 0.136318 0.26164 0.027159

16 0.0050252 15 0.131188 0.26124 0.027145

17 0.0048559 16 0.126163 0.26176 0.027148

18 0.0047670 17 0.121307 0.26167 0.027115

19 0.0043949 18 0.116540 0.26428 0.027209

20 0.0043159 20 0.107751 0.26039 0.027239

21 0.0037209 21 0.103435 0.25845 0.027100

22 0.0034477 22 0.099714 0.25540 0.027173

23 0.0030182 23 0.096266 0.25543 0.027241

24 0.0027663 24 0.093248 0.25672 0.027309

25 0.0023746 25 0.090482 0.25675 0.027064

26 0.0022350 26 0.088107 0.26072 0.027131

27 0.0020943 27 0.085872 0.26165 0.027241

28 0.0020141 28 0.083778 0.26010 0.027235

29 0.0020000 29 0.081764 0.26115 0.027257

From the above table we can se the cp value and the corresponding errors.

Here from the above table we have to find the lowest **xerror** from the above table which is row 10 having **xerror** of 0.24158 andhaving cp value of **0.0102805** and choose this value for Pruning the trees.

**Pruning**

After pruning the tree with cp value of 0.0102805 below is the decision tree split.

A picture containing text, map

Description automatically generated

Hence we will choose the Decision Tree Model having cp value of **0.0102805**

**Python:**

Used GridSearchCV to evaluate the best parameters for the DecisionTreeRegressor

**Random Forest**

In order to evaluate the Random forest model we will have to figure out the best parameters to tune the model.

Random forest chooses a random subset of features and builds many Decision Trees. The model averages out all the predictions of the Decisions trees.

Random forest has some parameters that can be changed to improve the generalization of the prediction.

To perform tuning in Random forest model, there are lot of combination possible between the parameters. We need not necessarily have the time to try all of them. A good alternative is to let the machine find the best combination for you. There are two methods available:

Random Search CV

Grid Search CV

* **Random Search CV**

The big difference between random search and grid search is, random search will not evaluate all the combination of hyperparameter in the searching space. Instead, it will randomly choose combination at every iteration. The advantage is it lowers the computational cost.

* **Grid Search CV**

The grid search method is simple, the model will be evaluated over all the combination you pass in the function, using cross-validation.

For instance, you want to try the model with 10, 20, 30 number of trees and each tree will be tested over a number of mtry equals to 1, 2, 3, 4, 5. Then the machine will test 15 different models

One shortcoming of the grid search is the number of experimentations. It can become very easily explosive when the number of combinations are high. To overcome this issue, we can use the random search.

Below steps are performed for tuning the parameters in this category

Evaluate the model with the default setting

Find the best number of mtry

Find the best number of maxnodes

Find the best number of ntrees

Evaluate the model on the test dataset

1. Default model setting

mtry RMSE Rsquared MAE

2 1312.5155 0.7576485 1068.2367

30 699.0596 0.8694401 501.1746

58 717.0394 0.8618254 511.8403

RMSE was used to select the optimal model using the smallest value.

The final value used for the model was mtry = 30.

1. Find the best number for mtry

Random Forest

584 samples

11 predictor

No pre-processing

Resampling: Cross-Validated (10 fold)

Summary of sample sizes: 526, 524, 526, 526, 525, 527, ...

Resampling results across tuning parameters:

mtry RMSE Rsquared MAE

1 1644.9828 0.6899325 1338.6595

2 1313.4441 0.7552746 1068.9934

3 1112.6241 0.8019995 906.0369

4 991.8340 0.8297347 801.6068

5 915.9227 0.8458395 729.3193

6 849.9803 0.8572285 667.9831

7 818.0520 0.8612918 634.8530

8 787.1015 0.8670406 602.8755

9 768.7906 0.8693823 582.6770

10 748.6542 0.8720120 562.6374

11 735.8124 0.8730270 546.0716

12 725.1587 0.8737799 536.6704

13 717.4857 0.8737582 528.2280

14 708.9307 0.8748401 516.5062

15 704.6672 0.8749033 513.9898

16 704.0681 0.8733862 509.7422

17 700.9893 0.8740971 505.1333

18 699.4013 0.8734609 502.6266

19 698.8209 0.8727341 499.5897

20 702.5184 0.8704829 500.4494

21 698.5129 0.8712796 498.0443

22 696.1308 0.8716213 497.5672

23 697.0811 0.8711022 494.6835

24 697.9728 0.8701718 497.3992

25 696.7019 0.8704475 493.0754

26 699.0131 0.8693039 495.0060

27 700.7620 0.8682866 495.8567

28 698.3574 0.8689944 494.4803

29 696.7655 0.8694077 491.9764

30 696.2364 0.8696125 493.3337

31 702.6675 0.8671094 495.7274

32 703.1197 0.8667459 496.4222

33 705.1036 0.8656203 496.6865

34 703.1610 0.8663728 495.7021

35 704.6363 0.8656573 497.4559

36 706.3229 0.8648133 497.4911

37 706.3242 0.8647841 497.5231

38 708.8579 0.8638011 499.3306

39 706.4464 0.8648815 496.6333

40 710.1677 0.8632198 498.7343

41 710.5331 0.8629532 498.3275

42 713.9974 0.8615060 500.6013

43 708.9201 0.8636218 498.3882

44 712.2910 0.8621452 499.7501

45 712.2295 0.8624771 500.2021

46 711.7782 0.8624083 499.1182

47 713.0327 0.8619074 502.6282

48 717.5721 0.8602217 503.2950

49 717.4962 0.8602311 503.9455

50 716.9116 0.8604145 503.2830

RMSE was used to select the optimal model using the smallest value.

The final value used for the model was mtry = 22.

1. Search the best maxnodes

Call:

summary.resamples(object = results\_mtry)

Models: 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

Number of resamples: 10

MAE

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

5 767.2439 844.3854 881.8628 893.4081 950.0542 1011.8081 0

6 728.5348 802.7738 832.3388 846.7546 908.0112 965.2665 0

7 685.0401 758.8607 780.8687 795.6209 856.4272 897.0972 0

8 662.9237 712.1205 755.4608 771.1031 842.5189 885.0296 0

9 630.7817 687.9016 736.3151 745.3344 823.7177 861.6296 0

10 609.8500 671.0803 709.4413 723.3639 794.5073 836.1157 0

11 579.4238 640.8916 697.3121 700.5111 780.5448 803.4314 0

12 568.2763 624.1297 675.6337 684.9157 765.9041 784.6390 0

13 549.6237 602.1524 662.7656 669.7732 749.0397 786.6011 0

14 532.6342 586.5015 655.6709 658.1894 738.3890 759.5232 0

15 521.8767 585.6919 644.8458 648.2590 726.9984 755.3417 0

RMSE

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

5 1008.9769 1048.5081 1086.8177 1117.7909 1208.7104 1245.167 0

6 952.1401 990.3489 1031.0905 1061.2714 1148.7716 1194.440 0

7 884.5774 938.3794 966.8690 1002.6332 1081.3448 1165.070 0

8 864.6129 889.9078 940.8630 974.6535 1056.1175 1134.254 0

9 830.0750 874.5162 923.9911 952.7372 1039.9850 1117.121 0

10 811.8039 861.2995 892.9834 931.2488 1011.3682 1101.876 0

11 770.4974 827.1802 875.6839 904.6765 983.2377 1092.258 0

12 765.0322 807.5566 853.4762 884.0321 958.7085 1065.898 0

13 740.5182 787.5777 842.8451 868.9369 951.1517 1046.681 0

14 727.9616 770.6161 833.0966 854.0163 924.9364 1038.110 0

15 711.1475 770.6545 823.7311 846.0749 920.3967 1028.765 0

Rsquared

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

5 0.6938554 0.7097614 0.7518089 0.7585491 0.8070029 0.8402337 0

6 0.7146638 0.7326745 0.7756685 0.7800723 0.8256283 0.8620786 0

7 0.7138519 0.7641704 0.7979758 0.7971726 0.8408936 0.8634522 0

8 0.7262841 0.7694146 0.7960835 0.8001923 0.8443910 0.8695919 0

9 0.7296168 0.7763173 0.8059981 0.8062026 0.8440704 0.8769515 0

10 0.7363323 0.7940797 0.8156292 0.8138091 0.8470348 0.8789010 0

11 0.7353137 0.7960881 0.8306161 0.8200487 0.8535501 0.8824569 0

12 0.7487243 0.8037201 0.8315666 0.8252376 0.8599261 0.8897019 0

13 0.7535029 0.8140656 0.8369013 0.8307844 0.8633344 0.8891626 0

14 0.7571202 0.8173883 0.8412158 0.8342090 0.8677016 0.8913225 0

15 0.7595347 0.8238428 0.8411904 0.8353473 0.8646370 0.8916860 0

The RMSE value is lowest for the last variable of maxnode of 15 in the range of 5 to 15 hence we are increasing the range to see if we can get lower RMSE value.

Call:

summary.resamples(object = results\_mtry)

Models: 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30

Number of resamples: 10

MAE

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

20 473.5321 542.4091 608.7298 606.3256 673.7710 709.3608 0

21 476.8914 547.7003 597.3161 603.8443 673.6465 712.1251 0

22 476.6093 540.9525 591.1914 597.8982 668.0495 704.3189 0

23 463.6022 530.5021 582.4420 588.5547 653.4367 698.0927 0

24 457.1817 535.0028 580.7599 585.5922 650.4856 688.4335 0

25 456.0221 520.7667 574.9988 579.3660 643.2247 689.5704 0

26 451.4377 518.8795 565.6460 574.8158 642.2175 681.3772 0

27 454.0252 517.4506 557.4360 570.6345 630.5017 684.9210 0

28 447.1910 510.7210 565.6912 570.1022 636.3198 679.6027 0

29 446.6295 508.8869 561.9185 566.2376 627.1588 672.9389 0

30 447.3482 509.3766 551.4700 560.4073 623.5918 671.3699 0

RMSE

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

20 656.4053 725.6799 786.1218 801.8118 873.4118 1003.2926 0

21 647.9193 730.2291 775.4453 797.8269 874.1999 1006.5765 0

22 641.9152 726.0315 772.4805 793.0873 867.7069 990.2885 0

23 628.3417 710.3051 762.6889 783.8156 860.6296 1000.5604 0

24 623.6116 712.9738 759.8942 777.6351 853.0748 975.2144 0

25 618.8293 702.6266 756.1986 771.8145 850.0941 975.7519 0

26 613.8262 700.4286 749.3758 768.0384 845.7239 979.6350 0

27 608.1555 690.3085 733.0065 761.0021 845.6089 967.9286 0

28 604.7991 693.5567 738.4329 763.1449 841.6818 975.5228 0

29 609.8586 686.4099 737.4503 757.8185 839.2188 949.4342 0

30 585.2249 683.6303 723.3760 751.2537 833.4804 964.9042 0

Rsquared

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

20 0.7663880 0.8343571 0.8502591 0.8456571 0.8775908 0.9006987 0

21 0.7636186 0.8403882 0.8491267 0.8467718 0.8751139 0.9014486 0

22 0.7708685 0.8410813 0.8506710 0.8483681 0.8773395 0.9008521 0

23 0.7632178 0.8449581 0.8536833 0.8506556 0.8815792 0.9047388 0

24 0.7774760 0.8449893 0.8548502 0.8527845 0.8810237 0.9051461 0

25 0.7757244 0.8483441 0.8567934 0.8542195 0.8819494 0.9072864 0

26 0.7749198 0.8475756 0.8567235 0.8545334 0.8844455 0.9074433 0

27 0.7790511 0.8503781 0.8614487 0.8573947 0.8859362 0.9082261 0

28 0.7754985 0.8480422 0.8578318 0.8552612 0.8843738 0.9087051 0

29 0.7869723 0.8497905 0.8598691 0.8578209 0.8879039 0.9067494 0

30 0.7798420 0.8542507 0.8632330 0.8596317 0.8901541 0.9148010 0

The lowest RMSE value is obtained with a value of maxnode equals to 29.

1. Search the best ntrees

We will evaluate the model against various different values of ntree values and check for the lowest RMSE value

Call:

summary.resamples(object = results\_tree)

Models: 300, 350, 400, 450, 500, 600, 700, 800, 900, 1000, 2000

Number of resamples: 10

MAE

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

300 471.8031 529.2315 550.0352 563.5637 563.5476 701.3725 0

350 473.2556 529.5130 553.4704 564.2234 563.1298 701.4056 0

400 472.3819 530.8003 552.5291 563.6054 563.1972 698.2702 0

450 473.5425 530.9237 552.4987 563.8292 563.4603 695.8226 0

500 476.4971 529.3177 554.0157 564.5065 562.4558 697.5778 0

600 477.5289 526.8297 555.7179 564.4343 565.0917 697.2961 0

700 479.4249 528.4699 554.1740 565.0236 564.9340 699.5532 0

800 479.5105 527.8753 552.8420 564.2275 564.5558 700.4746 0

900 478.3937 526.9103 551.1343 563.3131 563.8625 699.6867 0

1000 477.9240 527.2542 551.3239 563.5925 564.1573 701.4939 0

2000 472.7829 525.7284 554.1856 563.4746 563.5521 700.4483 0

RMSE

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

300 637.8525 689.6740 726.6443 756.6863 773.0606 963.8417 0

350 641.2056 689.2481 727.5619 757.1019 773.2023 961.6620 0

400 642.3093 692.4700 726.8124 756.9874 774.1113 958.8572 0

450 641.1569 691.8682 728.2850 757.5729 775.9148 957.8094 0

500 645.4606 690.7793 729.3860 758.9659 777.6404 959.7363 0

600 645.5881 688.7110 730.5240 758.0282 778.4616 957.7675 0

700 646.3279 690.4180 731.1750 758.8117 778.1573 958.2026 0

800 646.8642 689.5627 732.2496 758.8308 777.4757 958.3157 0

900 646.7948 688.1940 731.7093 757.6967 775.8524 958.3160 0

1000 646.4904 688.8306 733.0021 758.1954 776.2241 960.2993 0

2000 646.1389 685.4945 735.3339 758.1196 776.8573 959.5625 0

Rsquared

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

300 0.8004121 0.8244392 0.8637908 0.8572290 0.8867574 0.9148434 0

350 0.7979128 0.8246993 0.8641937 0.8571644 0.8856815 0.9150534 0

400 0.7987052 0.8242603 0.8649707 0.8573889 0.8848235 0.9145247 0

450 0.7981755 0.8233254 0.8647852 0.8572060 0.8853972 0.9148847 0

500 0.7960215 0.8224836 0.8640643 0.8564846 0.8841993 0.9154010 0

600 0.7999719 0.8230333 0.8619222 0.8568596 0.8844493 0.9156211 0

700 0.7983891 0.8233773 0.8617278 0.8566021 0.8845881 0.9147283 0

800 0.7970091 0.8238261 0.8610294 0.8565735 0.8851346 0.9147592 0

900 0.7994256 0.8249282 0.8609721 0.8570156 0.8850398 0.9149694 0

1000 0.7992754 0.8249111 0.8603858 0.8568022 0.8849604 0.9148128 0

2000 0.7980774 0.8237666 0.8617882 0.8567690 0.8853807 0.9155520 0

From the above results, ntrees = 600 is having the lowest RMSE value hence model will be trained with this data.

**Python**:

Used RandomSearchCV for RandomForestRegressor to evaluate the best parameters for the Random Forest Model.

**4.Conclusion**

Comparing all the three models and plotting the measurement for comparison across the models gives us the below table.

**R Code:**

RMSE Rsquared MAE

Linear\_Regression 882.7007 0.8123687 606.5968

Decision\_Tree 1006.4888 0.7535465 740.4103

Random\_Forest 829.9320 0.8681302 641.1567

From the above Model comparison parameters we find that Random Forest Model is having the lowest RMSE hence we have used the Random Forest Model.

**Python:**

Below are the results of comparison of the Model.

| **Model Name** | **MAPE** | **RMSE** | **Rsquared** |
| --- | --- | --- | --- |
| **0** | Linear Regression | 18.570705 | 836.714760 | 0.787863 |
| **1** | Decision Tree | 23.053068 | 886.674832 | 0.761774 |
| **2** | Random Forest | 16.051256 | 642.591177 | 0.874879 |

Here also we find that the RMSE square is less for Random forest hence we have finalized the Random Forest Model.

Code for Python and R: Attached in the zip file.

**References**

1. For Data Cleaning and Model Development - <https://edwisor.com/career-data-scientist>

2. <https://stackoverflow.com/>