

Predicting Employee Absenteeism

Kishore Mohit

23-06-2018

CONTENT

S.No	Chapter	Page No.
1.	Introduction	3
	1.1 Problem Statement	3
	1.2 Data	3
2.	Methodology	6
	2.1 Data Pre-processing	6
	2.1.1 Missing Data	6
	2.1.2 Outlier Analysis	11
	2.1.3 Feature Selection	19
	2.2 Modeling	22
	2.2.1 Model selection	22
	2.2.2 Linear Regression	23
	2.2.3 Decision Tree Regression	24
	2.2.4 Random Forest	26
3.	Conclusion	28
	3.1 Model Evaluations	28
	3.2 Model Selection	29
	3.3 Results	29
	3.3.1 Changes in Company	29
	3.3.2 Loss Every Month Projection	30
4	Appendix A – R Code	32
5	Appendix B – Python Code	42

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

To test the data for number of genuine issue of absenteeism occur in a company. This project deal with finding an issue for absence so we can bring changes regarding the issue to reduce the number of absenteeism and if the absenteeism remains the same then how much loss we can project for coming year in every month.

1.2 Data

ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day
11	26.0	7.0	3	1	289.0	36.0	13.0	33.0	239554.0
36	0.0	7.0	3	1	118.0	13.0	18.0	50.0	239554.0
3	23.0	7.0	4	1	179.0	51.0	18.0	38.0	239554.0
7	7.0	7.0	5	1	279.0	5.0	14.0	39.0	239554.0
11	23.0	7.0	5	1	289.0	36.0	13.0	33.0	239554.0

Fig1.1 Absenteeism Sample Data (Columns 1-10)

Disciplinary failure	Education	Son	Social drinker	Social smoker	Pet	Weight	Height	Body mass index	Absenteeism time in hours
0.0	1.0	2.0	1.0	0.0	1.0	90.0	172.0	30.0	4.0
1.0	1.0	1.0	1.0	0.0	0.0	98.0	178.0	31.0	0.0
0.0	1.0	0.0	1.0	0.0	0.0	89.0	170.0	31.0	2.0
0.0	1.0	2.0	1.0	1.0	0.0	68.0	168.0	24.0	4.0
0.0	1.0	2.0	1.0	0.0	1.0	90.0	172.0	30.0	2.0

Fig1.2 Absenteeism Sample Data (Columns 12-21)

Employee Absenteeism

As you can see in the Figure below we have the following 21 variables, using which we have to correctly predict the Absence of employee:

Predictor Variable	
0	ID
1	Reason for absence
2	Month of absence
3	Day of the week
4	Seasons
5	Transportation expense
6	Distance from Residence to Work
7	Service time
8	Age
9	Work load Average/day
10	Hit target
11	Disciplinary failure
12	Education
13	Son
14	Social drinker
15	Social smoker
16	Pet
17	Weight
18	Height
19	Body mass index
20	Absenteeism time in hours

Fig 1.3 Predictor Variables

CHAPTER 2

METHODOLOGY

2.1 Data Pre-Processing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

Data goes through a series of steps during preprocessing:

- Data Cleaning: Data is cleansed through processes such as filling in missing values, smoothing the noisy data, or resolving the inconsistencies in the data.
- Data Integration: Data with different representations are put together and conflicts within the data are resolved.
- Data Transformation: Data is normalized, aggregated and generalized.
- Data Reduction: This step aims to present a reduced representation of the data in a data warehouse.
- Data Discretization: Involves the reduction of a number of values of a continuous attribute by dividing the range of attribute intervals.

2.1.1 Missing Data

Before starting with cleaning process in preprocessing, we have to impute the missing value using Mean, Median, KNN whichever suits better with the data and have more close to our prediction data.

For our dataset we are going to use Median to impute data because it's giving nearby value to our data value. Missing value in our dataset we can visualize in Python and R by plotting the missing as shown in Fig 2.1 below

Employee Absenteeism

```
plt.figure(figsize=(14,10))
sns.heatmap(Absent.isnull(),yticklabels=False,cbar=False,cmap='viridis')
<matplotlib.axes._subplots.AxesSubplot at 0x17e4683d860>
```

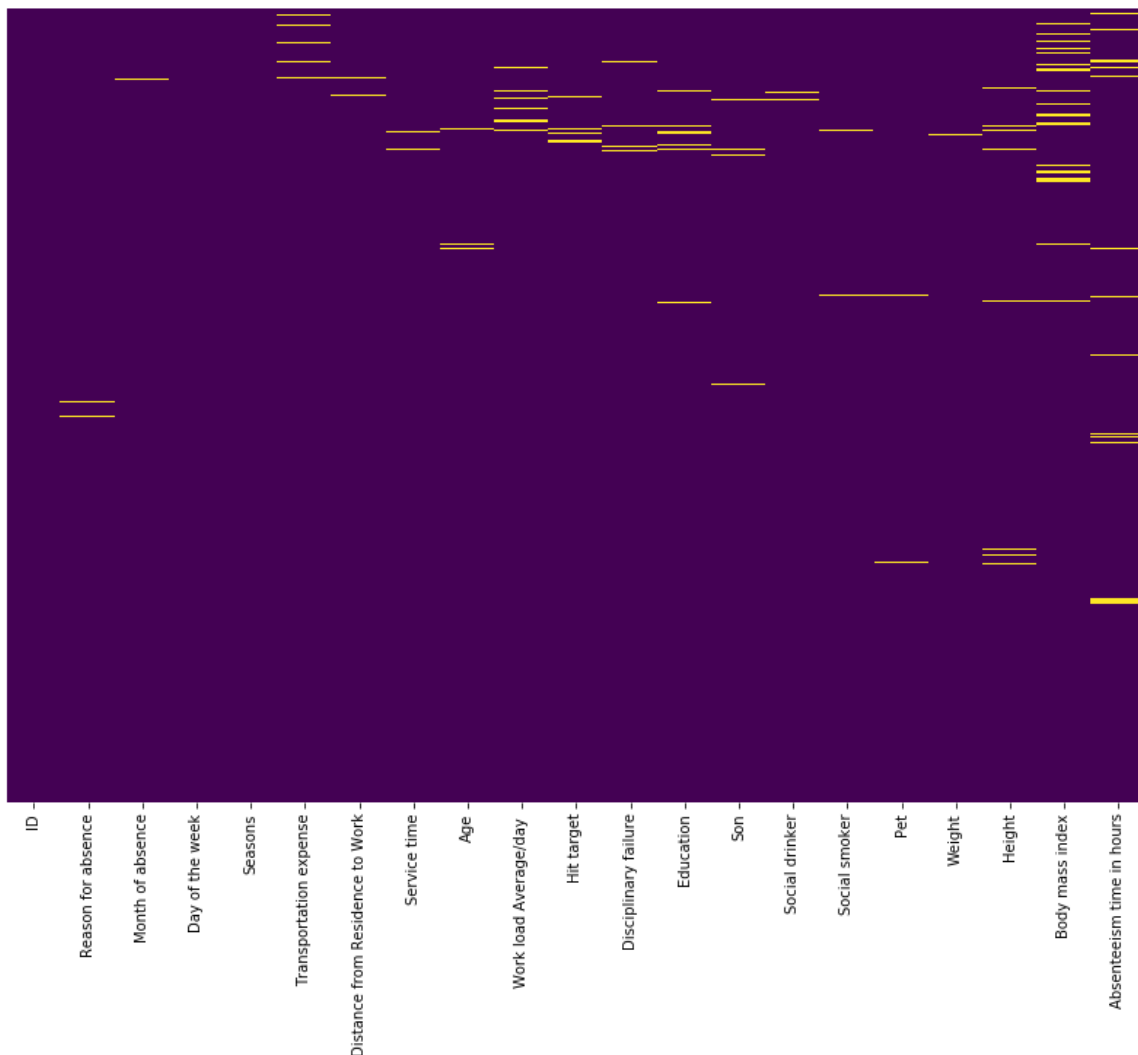


Figure 2.1. Missing Data Visualization in Python

Here we can see Yellow line in our diagram that represents the number of data missing in visualize way to understand it better and we can Body Mass Index and even out target variable have missing data and we can display the Percentage of data missing in our variables using Python and R as we can see the Data frame below in fig 2.2 showing the % of missing data in our Dataset.

Employee Absenteeism

	Variable	Missing %
0	Body mass index	4.189189
1	Absenteeism time in hours	2.972973
2	Height	1.891892
3	Work load Average/day	1.351351
4	Education	1.351351
5	Transportation expense	0.945946
6	Son	0.810811
7	Disciplinary failure	0.810811
8	Hit target	0.810811
9	Social smoker	0.540541
10	Age	0.405405
11	Reason for absence	0.405405
12	Service time	0.405405
13	Distance from Residence to Work	0.405405
14	Social drinker	0.405405
15	Pet	0.270270
16	Weight	0.135135
17	Month of absence	0.135135
18	Seasons	0.000000
19	Day of the week	0.000000
20	ID	0.000000

Figure 2.2 Missing Percentage (%) of Data in our Dataset.

After Analyzing the Missing value in our dataset, now we have to impute the data with best suited method statistics that are Mean, Median, KNN as we know

There are several methods to impute missing value for a data such as:

- **Mean:** Mean is first method where we can find a mean of the data as per the columns and impute the data to the missing value.

Employee Absenteeism

- **Median:** Median is other method to find the median of the data as per the columns and immune the data to the missing value.
- **Mode:** Median is other method to find the mode of the data as per the columns and immune the data to the missing value.
- **KNN Imputation:** KNN is an algorithm that is useful for matching a point with its closest k neighbors in a multi-dimensional space. It can be used for data that are continuous, discrete, ordinal and categorical which makes it particularly useful for dealing with all kind of missing data. The assumption behind using KNN for missing values is that a point value can be approximated by the values of the points that are closest to it, based on other variables.

But as our dataset is working good with Median, we are going to use Median to impute our missing data by method shown below.

```
Absent['Reason for absence'] = Absent['Reason for absence'].fillna(Absent['Reason for absence'].median())
```

```
Absent['Month of absence'] = Absent['Month of absence'].fillna(Absent['Month of absence'].median())
```

```
Absent['Transportation expense'] = Absent['Transportation expense'].fillna(Absent['Transportation expense'].median())
```

```
Absent['Distance from Residence to Work'] = Absent['Distance from Residence to Work'].fillna(Absent['Distance from Residence to Work'].median())
```

```
Absent['Service time'] = Absent['Service time'].fillna(Absent['Service time'].median())
```

```
Absent['Age'] = Absent['Age'].fillna(Absent['Age'].median())
```

```
Absent['Work load Average/day'] = Absent['Work load Average/day'].fillna(Absent['Work load Average/day'].median())
```

```
Absent['Hit target'] = Absent['Hit target'].fillna(Absent['Hit target'].median())
```

```
Absent['Disciplinary failure'] = Absent['Disciplinary failure'].fillna(Absent['Disciplinary failure'].median())
```

```
Absent['Education'] = Absent['Education'].fillna(Absent['Education'].median())
```

```
Absent['Son'] = Absent['Son'].fillna(Absent['Son'].median())
```

```
Absent['Social drinker'] = Absent['Social drinker'].fillna(Absent['Social drinker'].median())
```

```
Absent['Social smoker'] = Absent['Social smoker'].fillna(Absent['Social smoker'].median())
```


Employee Absenteeism

```
Absent['Pet']= Absent['Pet'].fillna(Absent['Pet'].median())
```

```
Absent['Weight']= Absent['Weight'].fillna(Absent['Weight'].median())
```

```
Absent['Height']= Absent['Height'].fillna(Absent['Height'].median())
```

```
Absent['Body mass index']= Absent['Body mass index'].fillna(Absent['Body mass index'].median())
```

```
Absent['Absenteeism time in hours']= Absent['Absenteeism time in hours'].fillna(Absent['Absenteeism time in hours'].median())
```

After Imputation now can we can visualize the data plot to see if our data is still missing or we have done imputing the missing data with Median that is shown below in fig 2.3.

```
<matplotlib.axes._subplots.AxesSubplot at 0x17e6a4cceb8>
```

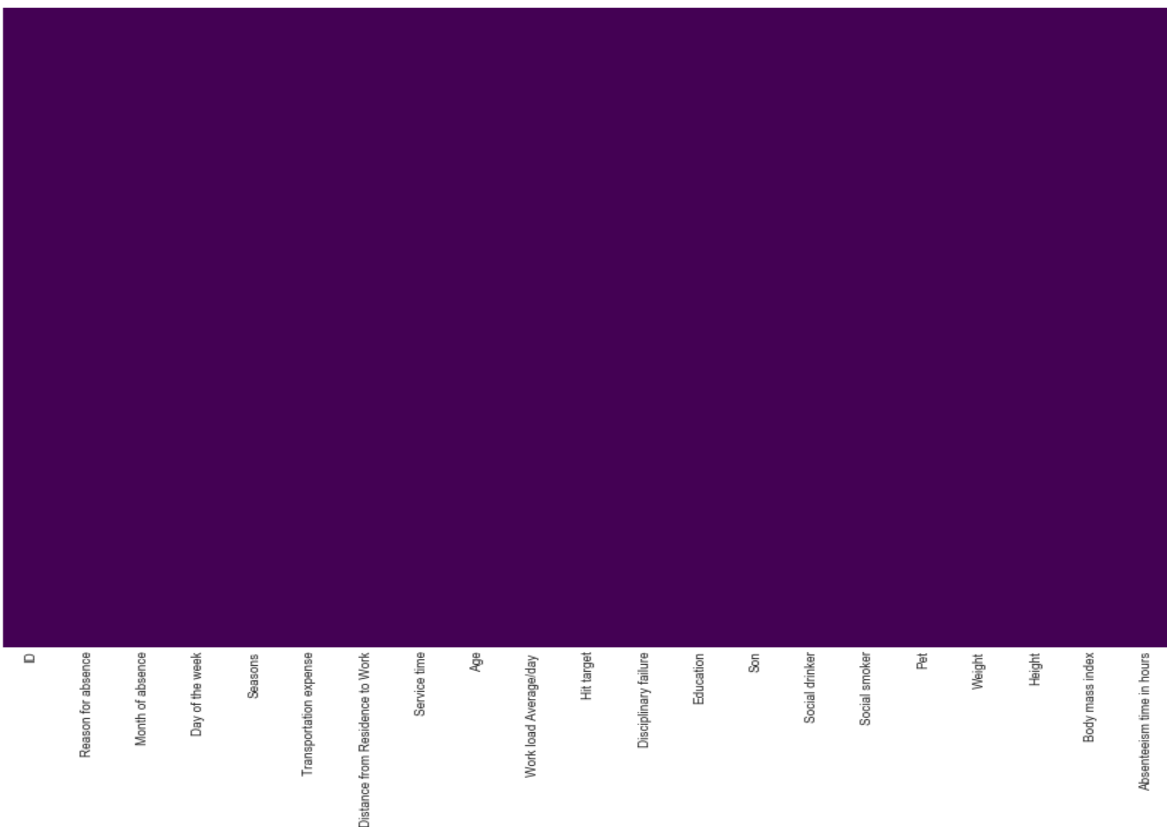


Figure 2.3 No Missing Value in Dataset after impute.

2.1.2. Outlier Analysis

For Outlier analysis we are going to visualize the Outliers present in data using Boxplot that is best method to represent the data with outliers graphically and helps us to define the quartile value as the same time. Box plot diagram is a graphical method typically depicted by quartiles and inter quartiles that helps in defining the upper limit and lower limit beyond which any data lying will be considered as outliers. *The very purpose of boxplot is to identify outliers and discard it from the data series* before making any further observation so that the conclusion made from the study gives more accurate results not influenced by any extremes or abnormal values.

A box plot is constructed by drawing a box between the upper and lower quartiles with a solid line drawn across the box to locate the median. The following quantities (called fences) are needed for identifying extreme values in the tails of the distribution:

lower inner fence: $Q1 - 1.5 \cdot IQ$

upper inner fence: $Q3 + 1.5 \cdot IQ$

lower outer fence: $Q1 - 3 \cdot IQ$

upper outer fence: $Q3 + 3 \cdot IQ$

A point beyond an inner fence on either side is considered a mild outlier. A point beyond an outer fence is considered an extreme outlier. Below is R code shown for analysis of Outlier using boxplot.

```
for (i in 1:length(Numerical))  
{  
  
  assign(paste0("gn",i),ggplot(aes_string(y=(Numerical[i]),x = "Absenteeism.time.in.hours"),  
data = subset(Absenteeism))+stat_boxplot(geom = "errorbar", width = 0.5) +
```

Employee Absenteeism

```
geom_boxplot(outlier.colour="red", fill = "blue", outlier.shape=18, outlier.size=1,  
notch=FALSE)+theme(legend.position="bottom")+labs(y=Numerical[i],x="Absenteeism.time.in.hours")+ggtitle(paste("Box plot for",Numerical[i]))
```

```
}
```

```
gridExtra::grid.arrange(gn1,gn2,gn3,ncol=3)
```

```
gridExtra::grid.arrange(gn4,gn5,gn6,ncol=3)
```

```
gridExtra::grid.arrange(gn7,gn8,gn9,ncol=3)
```

```
gridExtra::grid.arrange(gn10,gn11,ncol=2)
```

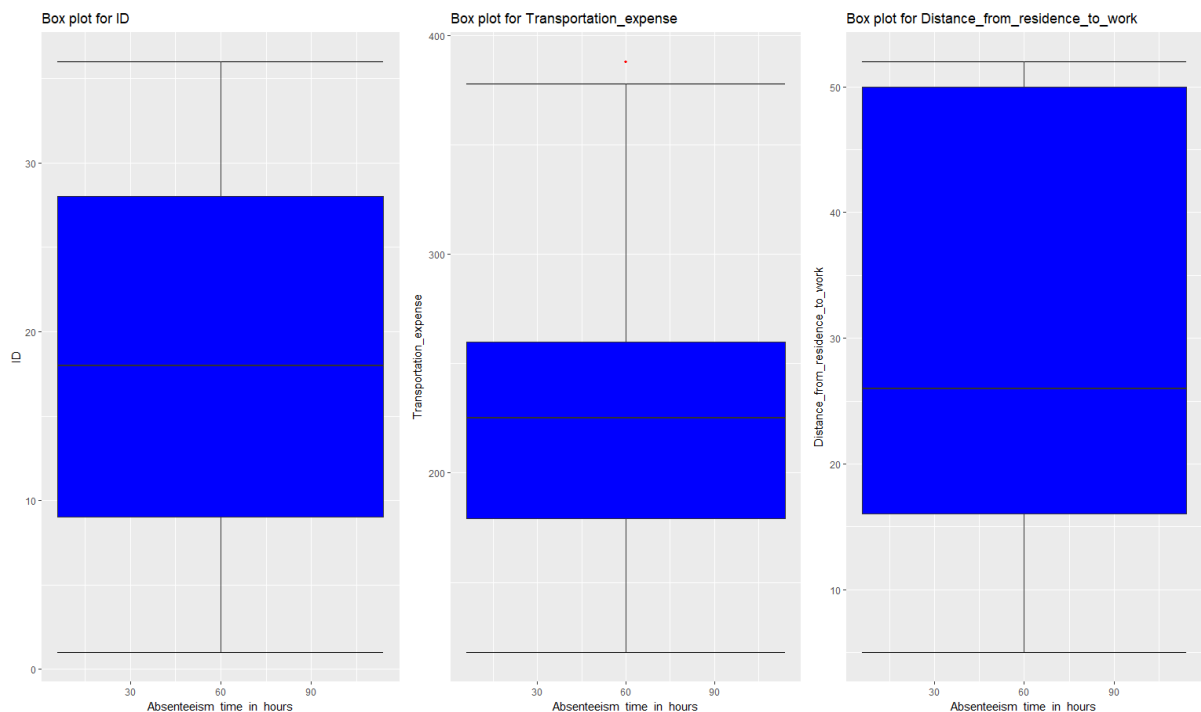


Figure 2.4 Outliers Box Plot and Outlier present in Transport_expense

Employee Absenteeism

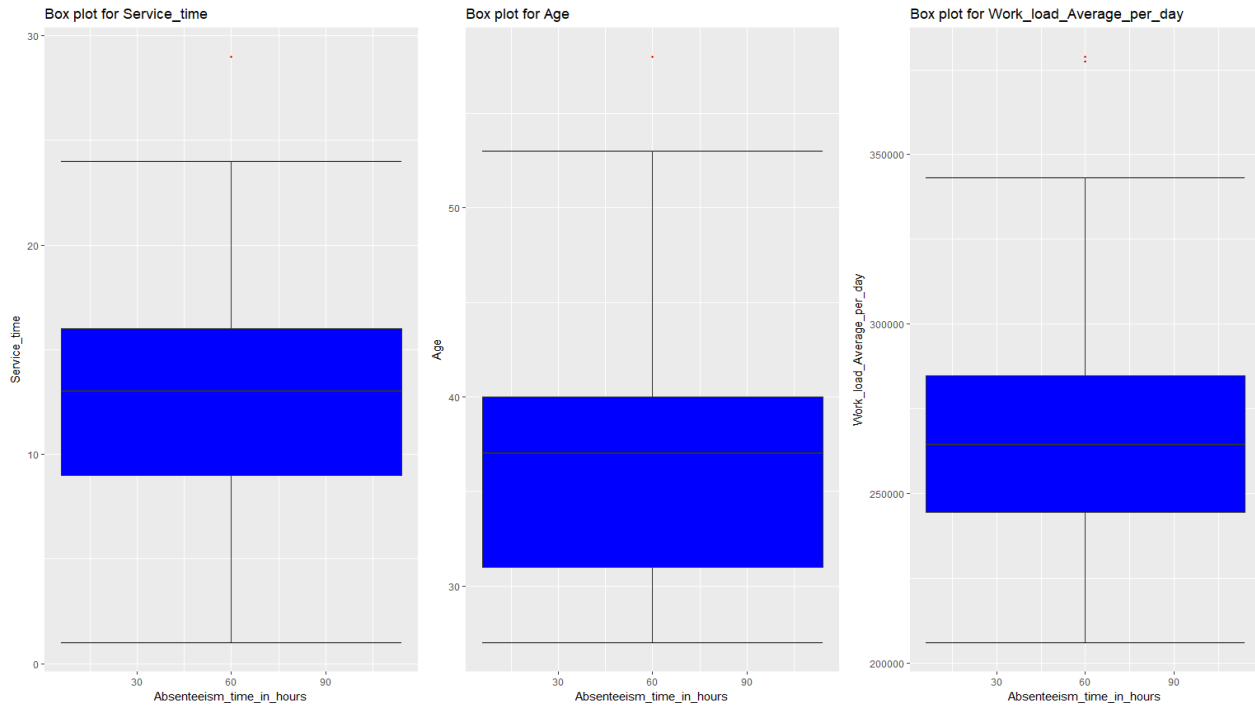


Figure 2.5 Outliers Box Plot and Outlier present in All Variable

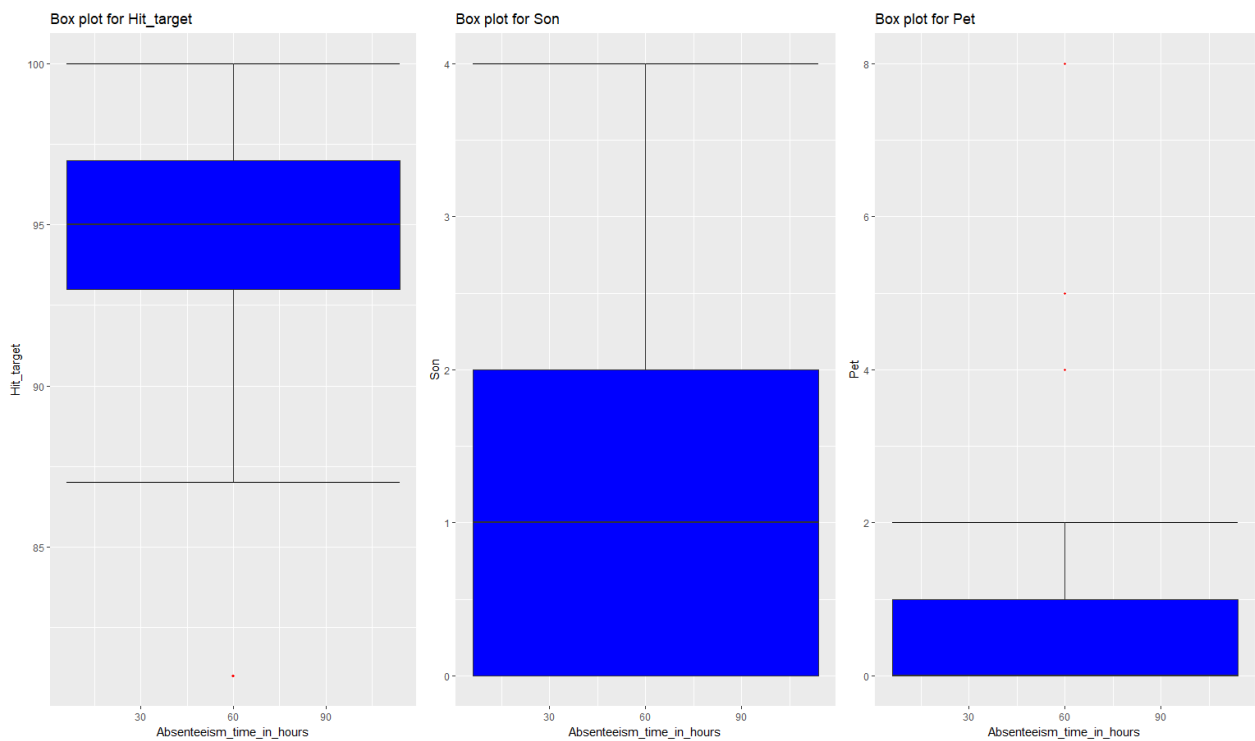


Figure 2.6 Outliers Box Plot and Outlier present in Hit_target and Pet

Employee Absenteeism

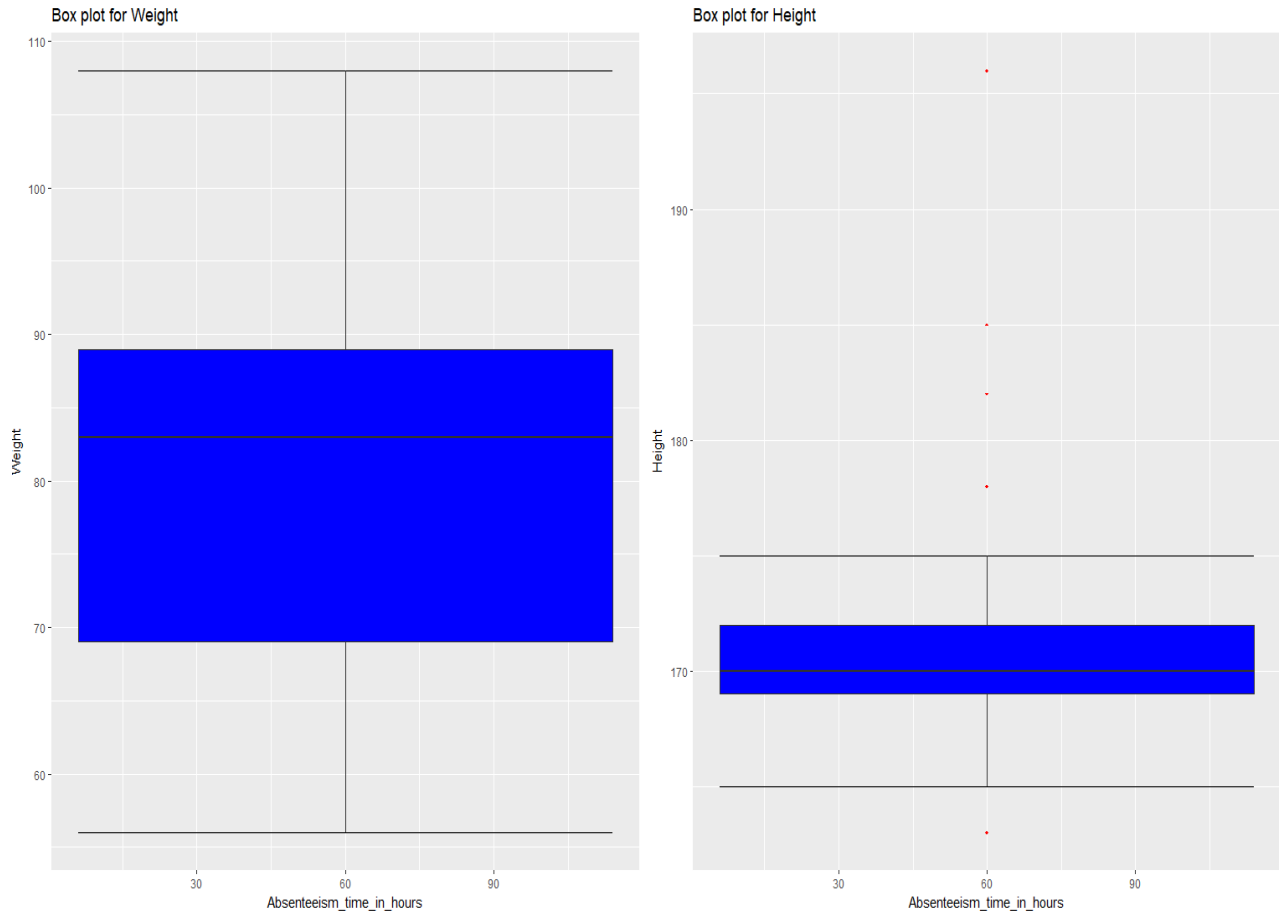


Figure 2.7 Outliers Box Plot and Outlier present in Height

Figure 2.8 Outliers Box Plot for our target Variable Absenteeism in Hours

Now we know we have outlier in most of our numerical variable so now we are going to remove those outliers and again impute the deleted outliers with median as Median turn out to be best for missing data impute for this data set and we will apply code written below to remove outliers and impute missing values.

```
Out = Absenteeism$`Transportation.expense`[Absenteeism$`Transportation.expense` %in%  
boxplot.stats(Absenteeism$`Transportation.expense`)$out]
```

```
Absenteeism$`Transportation.expense`[(Absenteeism$`Transportation.expense` %in% Out)] = NA
```

Employee Absenteeism

```
Absenteeism$`Transportation.expense`[is.na(Absenteeism$`Transportation.expense`)] =  
median.default(Absenteeism$`Transportation.expense`, na.rm = T)
```

```
Out = Absenteeism$`Service.time`[Absenteeism$`Service.time` %in% boxplot.stats(Absenteeism$`Service.time`)$out]
```

```
Absenteeism$`Service.time`[(Absenteeism$`Service.time` %in% Out)] = NA
```

```
Absenteeism$`Service.time`[is.na(Absenteeism$`Service.time`)] = median(Absenteeism$`Service.time`, na.rm = T)
```

```
Out = Absenteeism$Age[Absenteeism$Age %in% boxplot.stats(Absenteeism$Age)$out]
```

```
Absenteeism$Age[(Absenteeism$Age %in% Out)] = NA
```

```
Absenteeism$Age[is.na(Absenteeism$Age)] = median(Absenteeism$Age, na.rm = T)
```

```
Out = Absenteeism$`Work.load.Average.per.day`[Absenteeism$`Work.load.Average.per.day` %in%  
boxplot.stats(Absenteeism$`Work.load.Average.per.day`)$out]
```

```
Absenteeism$`Work.load.Average.per.day`[(Absenteeism$`Work.load.Average.per.day` %in% Out)] = NA
```

```
Absenteeism$`Work.load.Average.per.day`[is.na(Absenteeism$`Work.load.Average.per.day`)] =  
median(Absenteeism$`Work.load.Average.per.day`, na.rm = T)
```

```
Out = Absenteeism$`Hit.target`[Absenteeism$`Hit.target` %in% boxplot.stats(Absenteeism$`Hit.target`)$out]
```

```
Absenteeism$`Hit.target`[(Absenteeism$`Hit.target` %in% Out)] = NA
```

```
Absenteeism$`Hit.target`[is.na(Absenteeism$`Hit.target`)] = median(Absenteeism$`Hit.target`, na.rm = T)
```

```
Out = Absenteeism$Pet[Absenteeism$Pet %in% boxplot.stats(Absenteeism$Pet)$out]
```

```
Absenteeism$Pet[(Absenteeism$Pet %in% Out)] = NA
```

```
Absenteeism$Pet[is.na(Absenteeism$Pet)] = median(Absenteeism$Pet, na.rm = T)
```

```
Out = Absenteeism$Height[Absenteeism$Height %in% boxplot.stats(Absenteeism$Height)$out]
```

```
Absenteeism$Height[(Absenteeism$Height %in% Out)] = NA
```

```
Absenteeism$Height[is.na(Absenteeism$Height)] = median(Absenteeism$Height, na.rm = T)
```

```
Out = Absenteeism$Absenteeism.time.in.hours[Absenteeism$Absenteeism.time.in.hours %in%  
boxplot.stats(Absenteeism$Absenteeism.time.in.hours)$out]
```

Employee Absenteeism

```
Absenteeism$Absenteeism.time.in.hours[(Absenteeism$Absenteeism.time.in.hours %in% Out)] = NA
```

```
Absenteeism$Absenteeism.time.in.hours[is.na(Absenteeism$Absenteeism.time.in.hours)] =  
median(Absenteeism$Absenteeism.time.in.hours, na.rm = T)
```

After removing the outliers and imputing the data with median we will again analyze the outliers with Boxplot to see our outlier that we are able to overcome outliers or not.

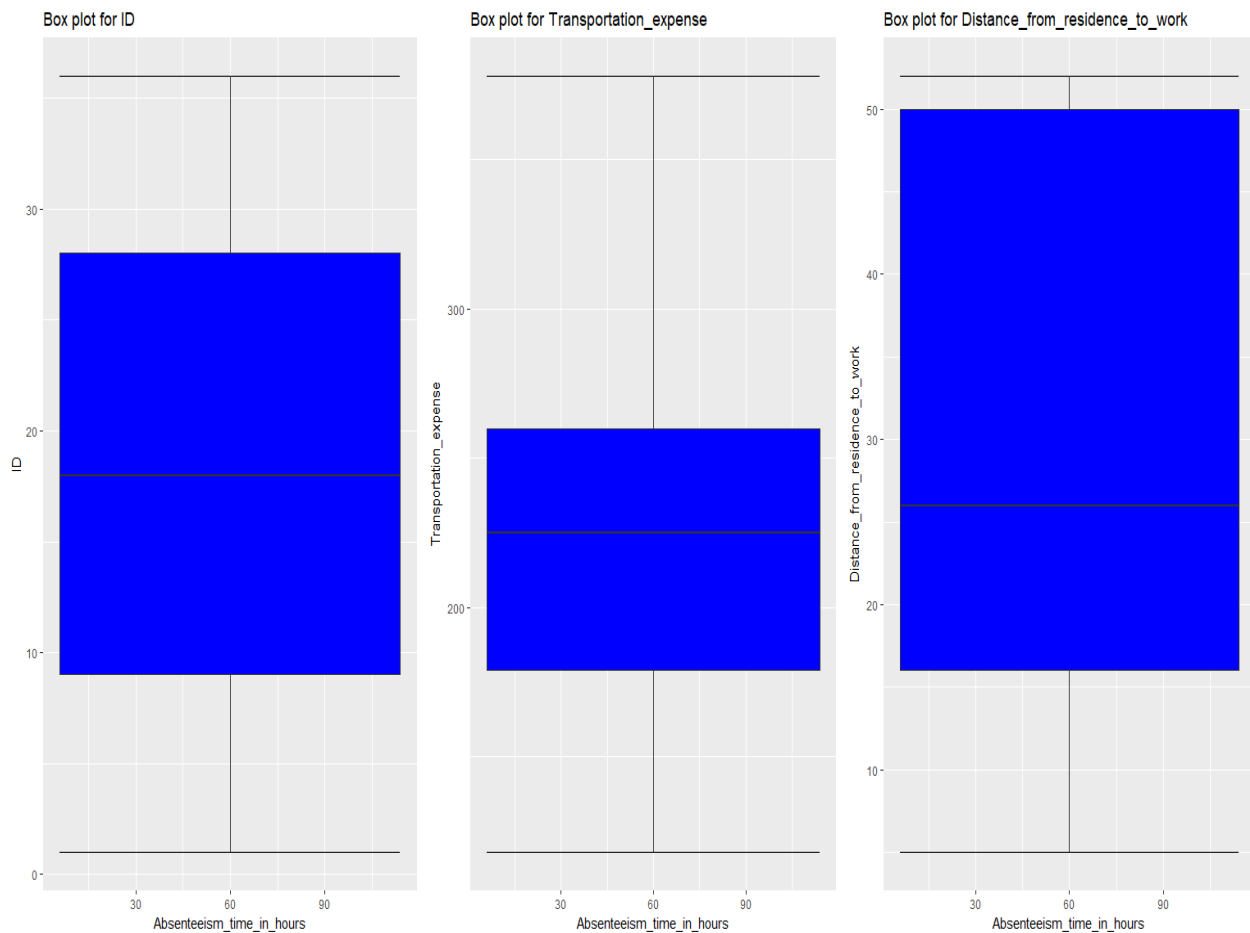


Figure 2.9 No Outliers Present after imputation in Box Plot

Employee Absenteeism

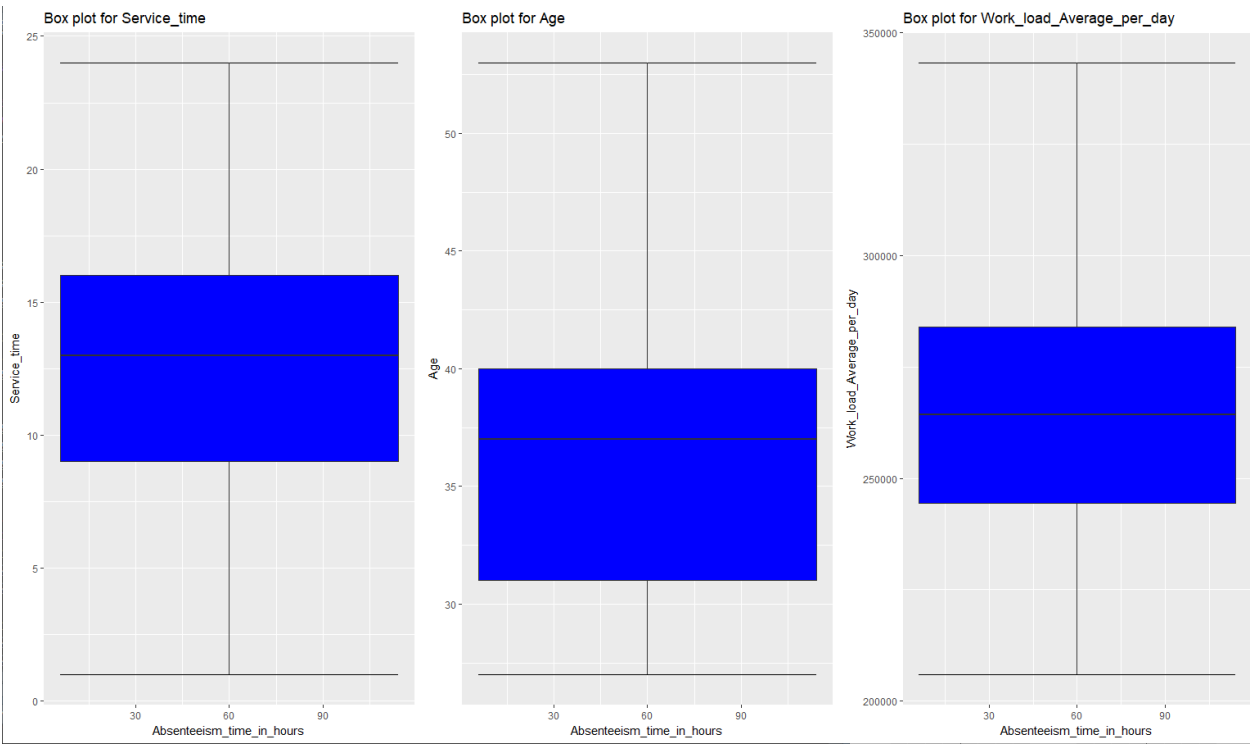


Figure 2.10 No Outliers Present after imputation in Box Plot

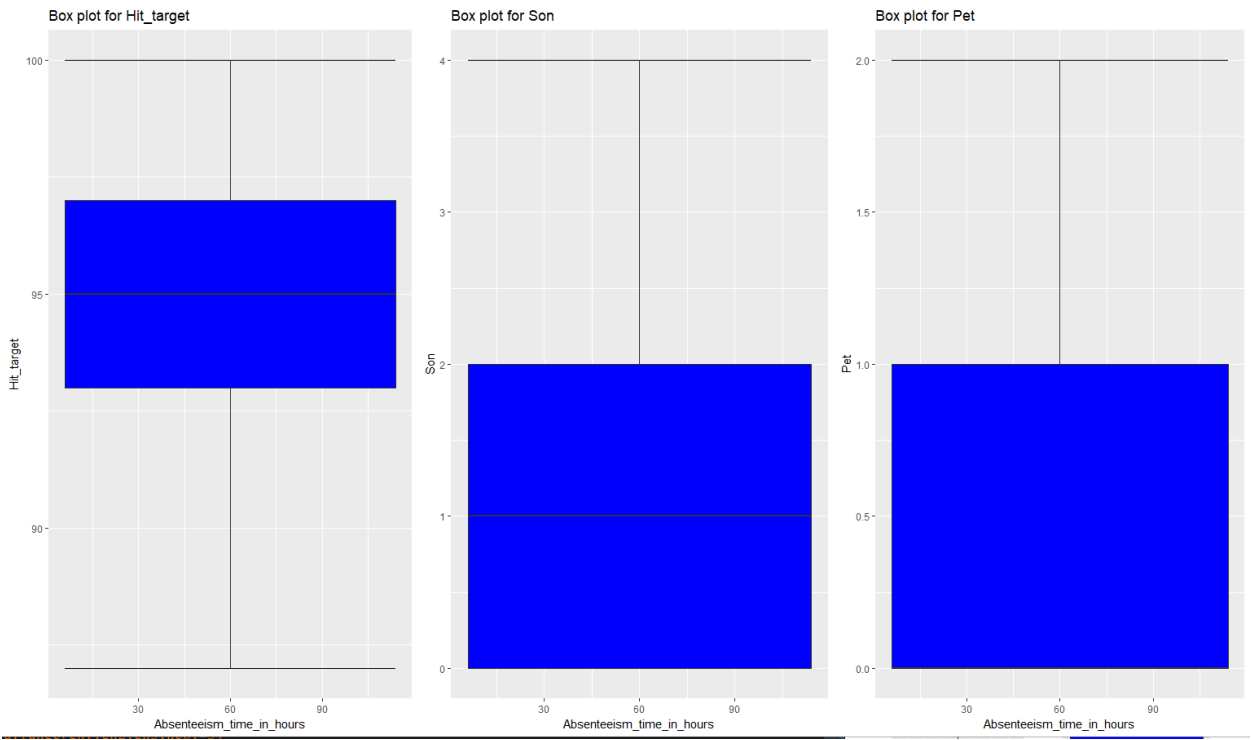


Figure 2.11 No Outliers Present after imputation in Box Plot

Employee Absenteeism

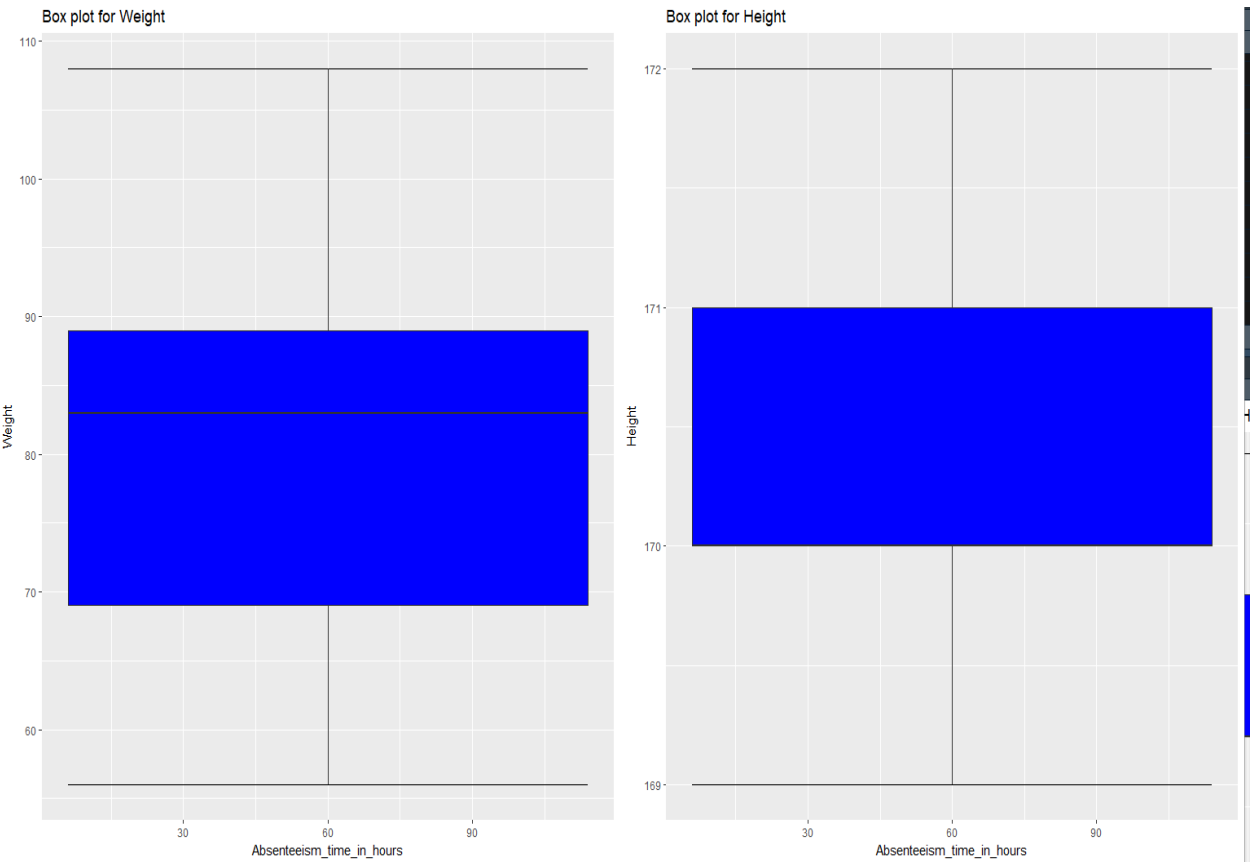


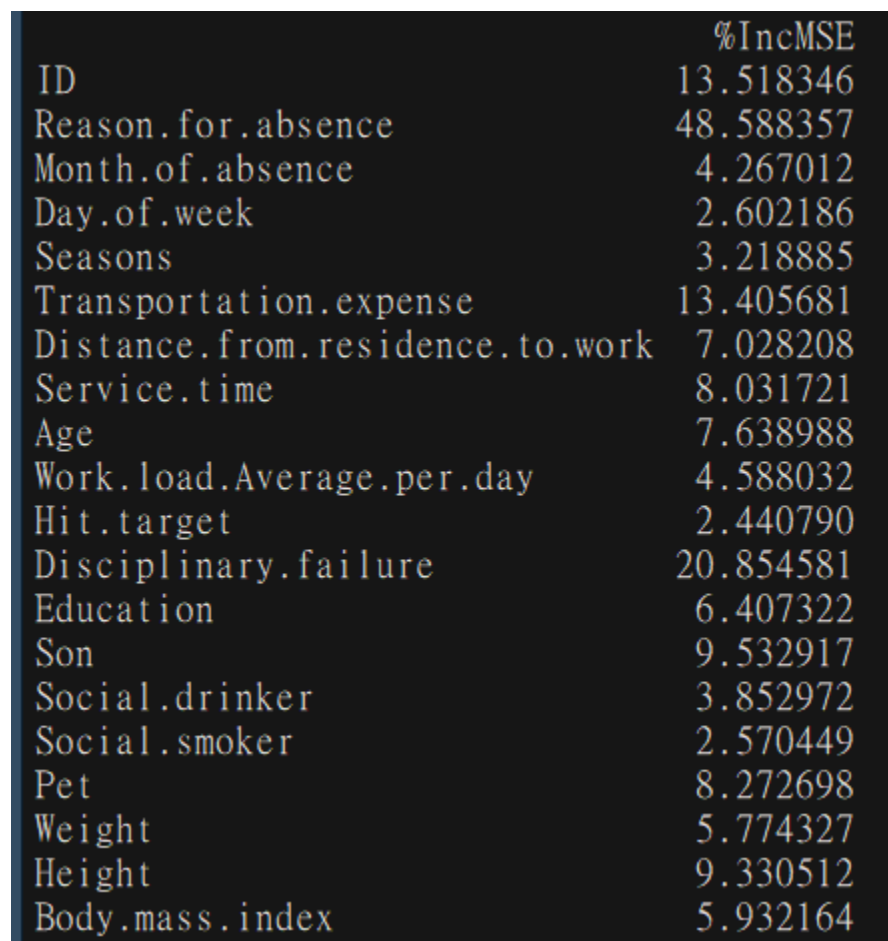
Figure 2.12 No Outliers Present after imputation in Box Plot

2.1.3 Feature Selection

Before performing any type of modeling, 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. There are several methods of doing that. Below we have used *Random Forests* to perform features selection.

```
Random.Model = randomForest(Absenteeism$Absenteeism.time.in.hours~.,data = Absenteeism ,
importance= TRUE,ntree = 500)

importance(Random.Model,type = 1)
```



	%IncMSE
ID	13.518346
Reason.for.absence	48.588357
Month.of.absence	4.267012
Day.of.week	2.602186
Seasons	3.218885
Transportation.expense	13.405681
Distance.from.residence.to.work	7.028208
Service.time	8.031721
Age	7.638988
Work.load.Average.per.day	4.588032
Hit.target	2.440790
Disciplinary.failure	20.854581
Education	6.407322
Son	9.532917
Social.drinker	3.852972
Social.smoker	2.570449
Pet	8.272698
Weight	5.774327
Height	9.330512
Body.mass.index	5.932164

Figure 2.13 Feature %IncMSE for every Variable

Employee Absenteeism

Now we need to choose most important features out of all the features for our further model development, so we will be using Random forest for that classification that with all the variable, our accuracy and after dropping unnecessary variable our accuracy, we will drop variable by method show below.

```
# Create a random forest classifier
clf = RandomForestClassifier(n_estimators=1000, random_state=0, n_jobs=-1)

# Train the classifier
clf.fit(X_train, y_train)

# Print the name and gini importance of each feature
for feature in zip(feats_labels, clf.feature_importances_):
    print(feature)

('Reason for absence', 0.12562309953318676)
('Month of absence', 0.08143380624380651)
('Day of the week', 0.15545751889649378)
('Seasons', 0.09044886841309066)
('Transportation expense', 0.02776480409230948)
('Distance from Residence to Work', 0.02445851438059047)
('Service time', 0.03100929699107872)
('Age', 0.029812615234451178)
('Work load Average/day ', 0.10626093376446596)
('Hit target', 0.08741854011426482)
('Disciplinary failure', 0.02908872076896042)
('Education', 0.006098773730622101)
('Son', 0.028072782548844263)
('Social drinker', 0.011828662374680495)
('Social smoker', 0.003731538051926149)
('Pet', 0.01293052108861812)
('Weight', 0.0353305111850389)
('Height', 0.038427387088001023)
('Body mass index', 0.07480310549956982)
```

Figure 2.14 All Feature Variable with Their value.

Now we will Perform accuracy test for full variable using Random Forest and will see the result.

```
# Applying The Full Featured Classifier To The Test Data
y_pred = clf.predict(X_test)

# Viewing The Accuracy Of Our Full Feature Model
accuracy_score(y_test, y_pred)

0.43243243243243246
```

Figure 2.15 Model Accuracy with Full Variable

Employee Absenteeism

Now as we have selected the Featured variable and again we going check the accuracy of our data after remvoing some of unnecessary variables ,to check how much it affect our model .

```
for feature_list_index in sfm.get_support(indices=True):  
    print(feat_labels[feature_list_index])
```

```
Reason for absence  
Month of absence  
Day of the week  
Seasons  
Service time  
Age  
Work load Average/day  
Hit target  
Disciplinary failure  
Weight  
Height  
Body mass index
```

Figure 2.16 Important Feature with Have Value > 0.29

```
# Apply The Full Featured Classifier To The Test Data  
y_important_pred = clf_important.predict(X_important_test)  
  
# View The Accuracy Of Limited Feature Model  
accuracy_score(y_test, y_important_pred)  
  
0.43243243243243246
```

Figure 2.17 Model Accuracy with Important Variable

As we can observe in our accuracy there is no literal change in our accuracy even after dropping some of the variables so we are not affecting our model and so we can continue with our Model development with these important variables.

2.2 Modeling

2.2.1 Model Selection

In our early preprocessing phase, we observed our Target variable “Absenteeism in hours” have 19 unique Hours in target and 19 variables is a problem of regression but Classification so we will be proceeding further with Regression Model to find best fitted model accuracy for our dataset.

```
Absent['Absenteeism time in hours'].unique()  
array([ 4,  0,  2,  3,  8, 40,  1,  7, 32,  5, 16, 24, 64,  
       56, 80, 120, 112, 104, 48], dtype=int64)
```

Figure 2.18. Unique value for target variable

The dependent variable is falling under continuous variable so If the dependent variable, in our case *Absenteeism in hour*, is continuous then only predictive analysis that we can perform is Regression, and dependent variable is Interval or Ratio the normal method is to do a **Regression** analysis. But the dependent variable we are dealing with is *Continuous*, for which regression can be done, because even though it's have 19 values but it is continuous starting from 0 to 120, these Continuous values have an order associated with them so We will start our model building from the simplest to more complex. Therefore, we use Multiple Linear Regression. Regression residuals must be normally distributed. A linear relationship is assumed between the dependent variable and the independent variables. The residuals are homoscedastic and approximately rectangular-shaped. Absence of multicollinearity is assumed in the model, meaning that the independent variables are not too highly correlated. At the center of the multiple linear regression analysis is the task of fitting a single line through a scatter plot. More specifically the multiple linear regression fits a line through a multi-dimensional space of data points. The simplest form has one dependent and two independent variables. The dependent variable may also be referred to as the outcome variable

Employee Absenteeism

or regress and the independent variables may also be referred to as the predictor variables or regressors.

2.2.2 Linear Regression

```
Call:
lm(formula = Absenteeism.time.in.hours ~ ., data = train_lm)

Residuals:
    Min       1Q   Median       3Q      Max
-21.602  -4.421  -1.648   0.808  112.145

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    24.86162    5.35796   4.640 4.31e-06 ***
Reason.for.absence -0.49341    0.08168  -6.040 2.75e-09 ***
Month.of.absence   0.14065    0.16116   0.873  0.38317
Transportation.expense 1.48197    2.81295   0.527  0.59851
Distance.from.residence.to.work -5.34110    1.95589  -2.731  0.00651 **
Service.time      9.74397    4.54980   2.142  0.03264 *
Age              -2.92446    3.48192  -0.840  0.40131
Work.load.Average.per.day -0.77865    2.38508  -0.326  0.74419
Disciplinary.failure -13.24145    2.77936  -4.764 2.40e-06 ***
Son               5.16096    2.39845   2.152  0.03183 *
Height           5.41042    3.35739   1.611  0.10762
Body.mass.index  -2.10754    3.14259  -0.671  0.50272
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 2.20. Linear Model Summary Coefficient.

After Implementing linear regression, we found our MSE value and depend on that we can analyze Error Rate and Accuracy and our Error and Accuracy shown as below:

```
> predictions_lm = predict(linear_model, test_lm[,1:11])
> Predicted_LM = data.frame(predictions_lm)
> Actual = test_lm[,12]
> Error = Actual - Predicted_LM
> RMSE(Error)
[1] 9.81463
```

Figure 2.21. Linear Model Summary Coefficient.

Now we have obtained Error Rate 9.81 % and Accuracy 90.19 % for my prediction with Linear Regression, i.e. pretty good for our Model prediction.

2.2.3 Decision Regression Tree

Now i tried to implement and use a different regression model to predict our Absenteeism target variable. I will use a regression tree to predict the values of our target variable. Where I will Decision Tree Regression using the method anova that is helpful when we have both continuous and categorical variable in our Dataset and I receive a pretty well accuracy using that but not good as Linear Regression.

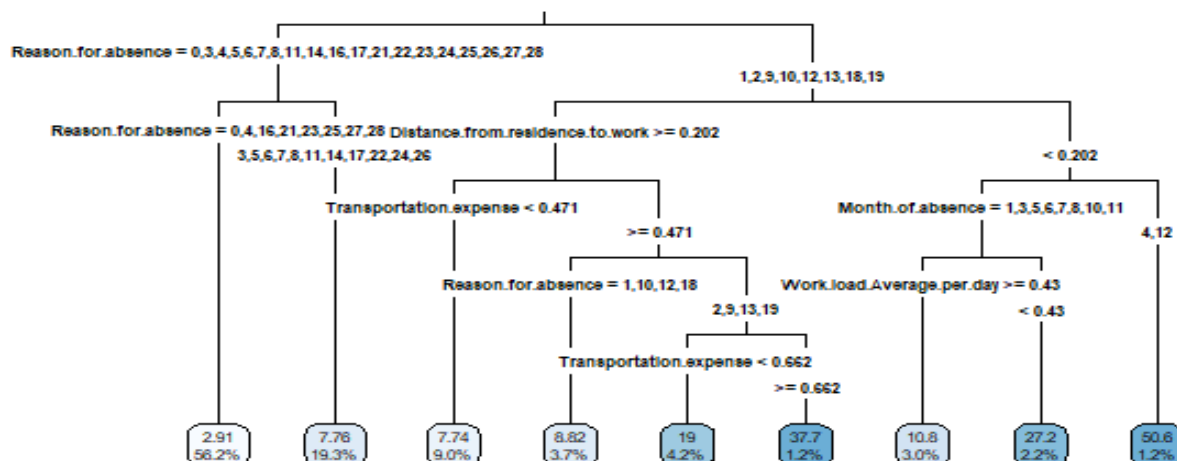


Figure 2.22. Regression tree for Absenteeism.

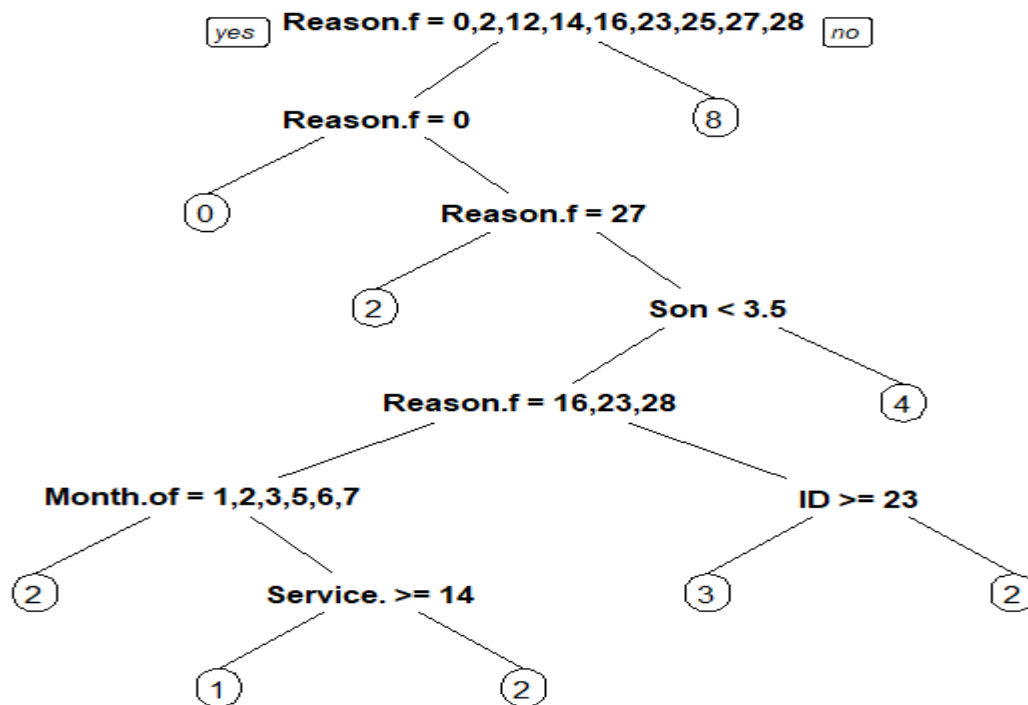


Figure 2.23. Tree on Reason of absence and Absent Hours.

After Implementing Tree regression, we found our MSE value and depend on that we can analyze Error Rate and Accuracy and our Error and Accuracy shown as below:

```

> dtree = rpart(Absenteeism.time.in.hours~.,data = train, method = "anova")
> dtree.plt = rpart.plot(dtree,type = 3,digits = 3,fallen.leaves = TRUE)
> prediction_dtree = predict(dtree,test[,-12])
> actual = test[,12]
> predicted_dtree = data.frame(prediction_dtree)
> Error = actual - predicted_dtree
> RMSE(Error)
[1] 13.4281
  
```

Figure 2.24. Error and Accuracy Rate with Decision Tree.

Decision Tree Performed Well but not as good as Linear Regression for our model and here we got Error Rate = 13.42% and Accuracy = 85.84% on our Prediction.

2.2.4. Random Forest Model

Now we going to implement and use a different Random Forest with different number of tree in random trees for better accuracy and we will observe how many number of tree is giving us better accuracy and fitting model to predict our Absenteeism target variable. I will tree in random forest as $n = 100$, $n = 500$, $n = 1000$ and we will check the accuracy if it is going to perform better than Linear Regression Model. As we know, Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks. In this post, you are going to learn, how the random forest algorithm works and several other important things about it. 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 100
> Random.Model = randomForest(Absenteeism.time.in.hours~.,train,importance = TRUE,ntree = 100)
> rand_pred = predict(Random.Model,test[-12])
> actual = test[,12]
> predicted_rand = data.frame(rand_pred)
> Error = actual - predicted_rand
> RMSE(Error)
[1] 12.28798
```

Figure 2.24. Error Rate with Decision Tree with N = 100.

Random Forest with $N = 100$ Performed Well and better than Decision tree but not as good as Linear Regression for our model and here we got Error Rate = 12.28% and Accuracy = 87.82% on our Prediction.

Employee Absenteeism

```
> # Random forest 500
> Random.Model = randomForest(Absenteeism.time.in.hours~.,train,importance = TRUE,ntree = 500)
> rand_pred = predict(Random.Model,test[-12])
> actual = test[,12]
> predicted_rand = data.frame(rand_pred)
> Error = actual - predicted_rand
> RMSE(Error)
[1] 12.31768
```

Figure 2.24. Error Rate with Decision Tree with N = 500.

Random Forest with N = 500 Performed Well and went minor high regarding N = 100 but better than Decision tree but not as good as Linear Regression for our model and here we got Error Rate = 12.31% and Accuracy = 87.69% on our Prediction.

```
> Random.Model = randomForest(Absenteeism.time.in.hours~.,train,importance = TRUE,ntree = 1000)
> rand_pred = predict(Random.Model,test[-12])
> actual = test[,12]
> predicted_rand = data.frame(rand_pred)
> Error = actual - predicted_rand
> RMSE(Error)
[1] 12.37231
```

Figure 2.25. Error Rate with Decision Tree with N = 1000.

Random Forest with N = 1000 Performed Well and again went minor high in error rate regarding N = 100 and N = 5000 but still better than Decision tree but not as good as Linear Regression for our model and here we got Error Rate = 12.37% and Accuracy = 87.62% on our Prediction.

CHAPTER 3

CONCLUSION

3.1. Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose.

There are several criteria that exist for evaluating and comparing models. We can compare the models using

any of the following criteria:

1. Predictive Performance
2. Interpretability
3. Computational Efficiency

In our case of Absenteeism Data, Interpretability and *Computation Efficiency*, do not hold much significance. Therefore, we will use *Predictive performance* as the criteria to compare and evaluate models. Predictive performance can be measured by comparing Predictions of the models with real values of the target variables and calculating some average error measure.

3.1.1 Root Mean Score Error (RMSE)

```
sample_lm = sample(1:nrow(Absenteeism),0.8*nrow(Absenteeism))
train_lm = Absenteeism[sample_lm,]
test_lm = Absenteeism[-sample_lm,]
linear_model = lm(Absenteeism.time.in.hours~.,data = train_lm)
summary(linear_model)
#vif(linear_model)
predictions_lm = predict(linear_model,test_lm[,1:11])
Predicted_LM = data.frame(predictions_lm)
Actual = test_lm[,12]
Error = Actual - Predicted_LM
RMSE(Error)
```

Figure 3.1. Implementation Of RMSE In Linear Regression

```
> predictions_lm = predict(linear_model, test_lm[,1:11])
> Predicted_LM = data.frame(predictions_lm)
> Actual = test_lm[,12]
> Error = Actual - Predicted_LM
> RMSE(Error)
[1] 9.516551
> |
```

Figure 3.2. RMSE In Linear Regression = 9.51

3.2. Model Selection

For our dataset Absenteeism Linear Regression performed so well with Error rate of 9.51% and Accuracy of 90.19% and where All other model Decision Tree achieve the Accuracy of 85.45% and Random Forest with N-tree = 100, 500, 1000 Obtained the Accuracy of 87.82%, 87.69% and 87.62% that is better than the Tree regression but not as good as Linear Regression Model, So for conclusion our Linear model performed so well in Predictive Performance so for our Dataset Absenteeism Linear Regression is best option to go with as our problem was related to regression.

3.3. Results

3.3.1 Changes in Company

As i chose to go with Linear Regression with important selected feature and we can get the coefficient related to our Important feature and can see which is affecting more on our model and which variable have negative significance with the coefficient and that is shown below:

Employee Absenteeism

```
coeff_df = pd.DataFrame(lm.coef_,Xtest.columns,columns=['Coefficient'])  
coeff_df
```

	Coefficient
Reason for absence	-0.338175
Month of absence	0.267631
Day of the week	-1.342719
Work load Average/day	0.000015
Hit target	0.060706
Height	0.207864
Body mass index	-0.318145
Pet	-0.439239
Distance from Residence to Work	-0.046633
Service time	0.231435
Age	0.007219

Figure 3. Coefficient value obtained from Linear Regression Model

As a conclusion Company need to work on their Work Load, Month, Age, BMI and check if person have pet or not if have then how many.

3.3.2. Loss Every Month Projection

WorkLoss

Work Load Loss/Month	
Janaury	5655839
Febraury	4719049
March	12141695
April	6757979
May	3475103
June	38827844
July	10088335
August	2763857
September	2430839
October	7740109
November	8614429
December	6535921

Fig.3.3 Work Loss Per Month at Work Load Basis

Employee Absenteeism

Absenteeism Evaluation Metrics

Here are three common evaluation metrics for Absenteeism problems:

****Absenteeism Time In HR**** (AHR)

****Work Load Avg/Day**** (WL)

****Service Time**** (ST)

****Month of Absence**** (MoA)

Calculating metrics:

****Work Loss/Month**** = $WL * AHR / ST$

****Loss Percent**** = $Work\ Loss\ Month / Work\ Load\ Month * 100$

Janaury	41.340797
Febraury	29.302331
March	62.263027
April	66.430274
May	48.004535
June	64.246266
July	86.919386
August	29.523785
September	32.641505
October	51.411054
November	57.308236
December	62.081930

dtype: float64 % Loss

Fig.3.4 Work Loss Per Month in Percentage % at Work Load Basis

Appendix A – R Code

```
#Required Libraries for our Dataset
```

```
x <- c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information", "MASS",  
"rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees', 'readxl')
```

```
#Load packages(x)
```

```
lapply(x, require, character.only = TRUE)
```

```
#Loading data
```

```
Absenteeism <- read_excel("Absenteeism.xls")
```

```
#Warning Option Off
```

```
options(warn = -1)
```

```
#Missing Data analysis In Percentage
```

```
missing <- data.frame(apply(Absenteeism,2,function(x){sum(is.na(x))}))
```

```
missing$Columns <- row.names(missing)
```

```
names(missing)[1] <- "Missing_PER"
```

```
missing$Missing <- (missing$Missing/nrow(Absenteeism)) * 100
```

```
missing <- missing[order(-missing$Missing),]
```

```
row.names(missing) <- NULL
```

```
missing <- missing[,c(2,1)]
```

```
#Plotting Histogram For Missing Data
```

```
ggplot(data = missing[1:8,], aes(x=reorder(Columns, -Missing_PER),y = Missing_PER))+geom_bar(stat = "identity",fill =  
"grey")+xlab("Parameter")+ggtitle("Missing data percentage") + theme_bw()
```

```
ggplot(data = missing[9:15,], aes(x=reorder(Columns, -Missing_PER),y = Missing_PER))+geom_bar(stat = "identity",fill =  
"grey")+xlab("Parameter")+ggtitle("Missing data percentage") + theme_bw()
```

```
ggplot(data = missing[16:21,], aes(x=reorder(Columns, -Missing_PER),y = Missing_PER))+geom_bar(stat = "identity",fill =  
"grey")+xlab("Parameter")+ggtitle("Missing data percentage") + theme_bw()
```

Employee Absenteeism

#Imputation Of Missing Data In Dataset Using Median

```
Absenteeism$'Reason for absence'[is.na(Absenteeism$'Reason for absence')] = median(Absenteeism$'Reason for absence', na.rm = T)
```

```
Absenteeism$'Month of absence'[is.na(Absenteeism$'Month of absence')] = median(Absenteeism$'Month of absence', na.rm = T)
```

```
Absenteeism$'Disciplinary failure'[is.na(Absenteeism$'Disciplinary failure')] = median(Absenteeism$'Disciplinary failure', na.rm = T)
```

```
Absenteeism$Education[is.na(Absenteeism$Education)] = median(Absenteeism$Education, na.rm = T)
```

```
Absenteeism$'Social drinker'[is.na(Absenteeism$'Social drinker')] = median(Absenteeism$'Social drinker', na.rm = T)
```

```
Absenteeism$'Social smoker'[is.na(Absenteeism$'Social smoker')] = median(Absenteeism$'Social smoker', na.rm = T)
```

```
Absenteeism$'Transportation expense'[is.na(Absenteeism$'Transportation expense')] =  
median.default(Absenteeism$'Transportation expense', na.rm = T)
```

```
Absenteeism$'Distance from Residence to Work'[is.na(Absenteeism$'Distance from Residence to Work')] =  
median(Absenteeism$'Distance from Residence to Work', na.rm = T)
```

```
Absenteeism$'Service time'[is.na(Absenteeism$'Service time')] = median(Absenteeism$'Service time', na.rm = T)
```

```
Absenteeism$Age[is.na(Absenteeism$Age)] = median(Absenteeism$Age, na.rm = T)
```

```
Absenteeism$'Work load Average/day'[is.na(Absenteeism$'Work load Average/day')] = median(Absenteeism$'Work load  
Average/day', na.rm = T)
```

```
Absenteeism$'Hit target'[is.na(Absenteeism$'Hit target')] = median(Absenteeism$'Hit target', na.rm = T)
```

```
Absenteeism$Son[is.na(Absenteeism$Son)] = median(Absenteeism$Son, na.rm = T)
```

```
Absenteeism$Pet[is.na(Absenteeism$Pet)] = median(Absenteeism$Pet, na.rm = T)
```

```
Absenteeism$Weight[is.na(Absenteeism$Weight)] = median(Absenteeism$Weight, na.rm = T)
```

```
Absenteeism$Height[is.na(Absenteeism$Height)] = median(Absenteeism$Height, na.rm = T)
```

```
Absenteeism$'Body mass index'[is.na(Absenteeism$'Body mass index')] = median(Absenteeism$'Body mass index', na.rm = T)
```

```
Absenteeism$'Absenteeism time in hours'[is.na(Absenteeism$'Absenteeism time in hours')] = median(Absenteeism$'Absenteeism  
time in hours', na.rm = T)
```

#Converting variables to their types as factor for categorical variable

```
Absenteeism$'Reason for absence' = as.factor(Absenteeism$'Reason for absence')
```


Employee Absenteeism

```
Absenteeism$'Month of absence' = as.factor(Absenteeism$'Month of absence')
```

```
Absenteeism$'Day of the week' = as.factor(Absenteeism$'Day of the week')
```

```
Absenteeism$Seasons = as.factor(Absenteeism$Seasons)
```

```
Absenteeism$'Disciplinary failure' = as.factor(Absenteeism$'Disciplinary failure')
```

```
Absenteeism$Education = as.factor(Absenteeism$Education)
```

```
Absenteeism$'Social drinker' = as.factor(Absenteeism$'Social drinker')
```

```
Absenteeism$'Social smoker' = as.factor(Absenteeism$'Social smoker')
```

```
#Changing Column Name As Per Need and For Better Understanding
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Absenteeism time in hours'] = 'Absenteeism.time.in.hours'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Reason for absence'] = 'Reason.for.absence'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Month of absence'] = 'Month.of.absence'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Day of the week'] = 'Day.of.week'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Transportation expense'] = 'Transportation.expense'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Distance from Residence to Work'] = 'Distance.from.residence.to.work'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Service time'] = 'Service.time'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Work load Average/day'] = 'Work.load.Average.per.day'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Hit target'] = 'Hit.target'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Disciplinary failure'] = 'Disciplinary.failure'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Social drinker'] = 'Social.drinker'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Social smoker'] = 'Social.smoker'
```

```
colnames(Absenteeism)[colnames(Absenteeism) == 'Body mass index'] = 'Body.mass.index'
```

```
#Outlier Analysis On our Numerical Variable
```

```
numeric_index = sapply(Absenteeism,is.numeric) #selecting only numeric
```

```
numerical = Absenteeism[,numeric_index]
```

Employee Absenteeism

```
Numerical = colnames(numerical)

#Plotting Box Plot For outlier Analysis in our Numerical Data

for (i in 1:length(Numerical))

{

  assign(paste0("gn",i), ggplot(aes_string(y = (Numerical[i]), x = "Absenteeism.time.in.hours"), data = subset(Absenteeism))+
  stat_boxplot(geom = "errorbar", width = 0.5) +geom_boxplot(outlier.colour="red", fill = "blue", outlier.shape=18,

  outlier.size=1, notch=FALSE)
+theme(legend.position="bottom")+labs(y=Numerical[i],x="Absenteeism.time.in.hours")+ggtitle(paste("Box plot
for",Numerical[i])))

}

#Plotting Generated Boxplot

gridExtra::grid.arrange(gn1,gn2,gn3,ncol=3)

gridExtra::grid.arrange(gn4,gn5,gn6,ncol=3)

gridExtra::grid.arrange(gn7,gn8,gn9,ncol=3)

gridExtra::grid.arrange(gn10,gn11,ncol=2)

#Outlier imputation using NA technique then Median for Missing Data

Out = Absenteeism$`Transportation.expense`[Absenteeism$`Transportation.expense` %in%
boxplot.stats(Absenteeism$`Transportation.expense`)$out]

Absenteeism$`Transportation.expense`[(Absenteeism$`Transportation.expense` %in% Out)] = NA

Absenteeism$`Transportation.expense`[is.na(Absenteeism$`Transportation.expense`)] =
median.default(Absenteeism$`Transportation.expense`, na.rm = T)

Out = Absenteeism$`Service.time`[Absenteeism$`Service.time` %in% boxplot.stats(Absenteeism$`Service.time`)$out]

Absenteeism$`Service.time`[(Absenteeism$`Service.time` %in% Out)] = NA

Absenteeism$`Service.time`[is.na(Absenteeism$`Service.time`)] = median(Absenteeism$`Service.time`, na.rm = T)

Out = Absenteeism$Age[Absenteeism$Age %in% boxplot.stats(Absenteeism$Age)$out]

Absenteeism$Age[(Absenteeism$Age %in% Out)] = NA
```

Employee Absenteeism

```
Absenteeism$Age[is.na(Absenteeism$Age)] = median(Absenteeism$Age, na.rm = T)
```

```
Out = Absenteeism$`Work.load.Average.per.day`[Absenteeism$`Work.load.Average.per.day` %in%  
boxplot.stats(Absenteeism$`Work.load.Average.per.day`)$out]
```

```
Absenteeism$`Work.load.Average.per.day`[(Absenteeism$`Work.load.Average.per.day` %in% Out)] = NA
```

```
Absenteeism$`Work.load.Average.per.day`[is.na(Absenteeism$`Work.load.Average.per.day`)] =  
median(Absenteeism$`Work.load.Average.per.day`, na.rm = T)
```

```
Out = Absenteeism$`Hit.target`[Absenteeism$`Hit.target` %in% boxplot.stats(Absenteeism$`Hit.target`)$out]
```

```
Absenteeism$`Hit.target`[(Absenteeism$`Hit.target` %in% Out)] = NA
```

```
Absenteeism$`Hit.target`[is.na(Absenteeism$`Hit.target`)] = median(Absenteeism$`Hit.target`, na.rm = T)
```

```
Out = Absenteeism$Pet[Absenteeism$Pet %in% boxplot.stats(Absenteeism$Pet)$out]
```

```
Absenteeism$Pet[(Absenteeism$Pet %in% Out)] = NA
```

```
Absenteeism$Pet[is.na(Absenteeism$Pet)] = median(Absenteeism$Pet, na.rm = T)
```

```
Out = Absenteeism$Height[Absenteeism$Height %in% boxplot.stats(Absenteeism$Height)$out]
```

```
Absenteeism$Height[(Absenteeism$Height %in% Out)] = NA
```

```
Absenteeism$Height[is.na(Absenteeism$Height)] = median(Absenteeism$Height, na.rm = T)
```

```
#Correlation plot on Numerical variable to remove most common correlated Data
```

```
corrgram(Absenteeism, order = F, upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")
```

```
#Making Copy of our Data
```

```
Absent = Absenteeism
```

```
Absenteeism = Absent
```

```
#Random Forest For Important Variables in Dataset
```

```
library(randomForest)
```

```
Random.Model = randomForest(Absenteeism$Absenteeism.time.in.hours~., data = Absenteeism, importance= TRUE, ntree = 500)
```

```
print(Random.Model)
```

```
attributes(Random.Model)
```

Employee Absenteeism

```
importance(Random.Model,type = 1)

varUsed(Random.Model) # to find which variables used in random forest

varImpPlot(Random.Model,sort = TRUE,n.var = 10,main="10 Imp. Variable")

varImp(Random.Model)

# Anova for categorical data to find important categorical variable for dataset

str(absent)

library(ANOVA.TFNs)

library(ANOVAreplication)

result =
aov(formula=Absenteeism.time.in.hours~Reason.for.absence+Month.of.absence+Day.of.week+Seasons+Disciplinary.failure+Education+Social.smoker+Social.drinker, data = Absenteeism)

summary(result)

#Decision Tree to Find Important Variable Participation in Dataset

library(rpart.plot)

tree <- rpart(Absenteeism.time.in.hours ~ . , method='class', data = Absenteeism)

printcp(tree)

plot(tree, uniform=TRUE, main="Main Title")

text(tree, use.n=TRUE, all=TRUE)

prp(tree)

#Removing Unnecessary and unimportant Variable from our Dataset

Absenteeism = subset(Absenteeism, select = -c(ID,Education,Day.of.week, Pet,Hit.target,
Seasons,Social.smoker,Social.drinker,Weight))

#Making New Set of Numerical column

Numerical_Col = c("Transportation.expense", "Age", "Son", "Height", "Body.mass.index", "Service.time", "Work.load.Average.per.day",

"Distance.from.residence.to.work")
```

Employee Absenteeism

```
#Applying Normalization to bring bring all data to same scale

for(i in Numerical_Col){

  print(i)

  Absenteeism[,i] = (Absenteeism[,i] - min(Absenteeism[,i]))/(max(Absenteeism[,i] - min(Absenteeism[,i])))

}

#Creating Sample Variable for Test and Train data for further Analysis

sample = sample(1:nrow(Absenteeism), 0.8 * nrow(Absenteeism))

train = Absenteeism[sample,]

test = Absenteeism[-sample,]

#Loading Import library require for tree analysis

library(mltools)

RMSE <- function(Error)

{

  sqrt(mean(Error^2))

}

#Implementing Decision Tree Model on dataset

dtree = rpart(Absenteeism.time.in.hours~.,data = train, method = "anova")

dtree.plt = rpart.plot(dtree,type = 3,digits = 3,fallen.leaves = TRUE)

prediction_dtree = predict(dtree,test[,12])

actual = test[,12]

predicted_dtree = data.frame(prediction_dtree)

Error = actual - predicted_dtree

RMSE(Error)

#summary(dtree)
```

Employee Absenteeism

```
#Error = 13.42 #Accuracy = 85.84
```

```
#Implementing Random forest Model on dataset with N = 100,500,1000
```

```
# Random forest 100
```

```
Random.Model = randomForest(Absenteeism.time.in.hours~.,train,importance = TRUE,ntree = 100)
```

```
rand_pred = predict(Random.Model,test[-12])
```

```
actual = test[,12]
```

```
predicted_rand = data.frame(rand_pred)
```

```
Error = actual - predicted_rand
```

```
RMSE(Error)
```

```
# Error rate = 12.28 # Accuracy 87.72
```

```
# Random forest 500
```

```
Random.Model = randomForest(Absenteeism.time.in.hours~.,train,importance = TRUE,ntree = 500)
```

```
rand_pred = predict(Random.Model,test[-12])
```

```
actual = test[,12]
```

```
predicted_rand = data.frame(rand_pred)
```

```
Error = actual - predicted_rand
```

```
RMSE(Error)
```

```
# Error rate = 12.31 # Accuracy 87.69
```

```
# Random forest 1000
```

```
Random.Model = randomForest(Absenteeism.time.in.hours~.,train,importance = TRUE,ntree = 1000)
```

```
rand_pred = predict(Random.Model,test[-12])
```

```
actual = test[,12]
```

```
predicted_rand = data.frame(rand_pred)
```

```
Error = actual - predicted_rand
```

Employee Absenteeism

```
RMSE(Error)

summary(Random.Model)

# Error rate = 12.37 # Accuracy 87.63

#Converting Categorical to numerical for linear Regression Analysis

Absenteeism$Reason.for.absence=as.numeric(Absenteeism$Reason.for.absence)

Absenteeism$'Month.of.absence' = as.numeric(Absenteeism$'Month.of.absence')

Absenteeism$'Disciplinary.failure' = as.numeric(Absenteeism$'Disciplinary.failure')

#Implementing Linear Regression Model on dataset with new Sample

sample_lm = sample(1:nrow(Absenteeism),0.8*nrow(Absenteeism))

train_lm = Absenteeism[sample_lm,]

test_lm = Absenteeism[-sample_lm,]

linear_model = lm(Absenteeism.time.in.hours~.,data = train_lm)

summary(linear_model)

#vif(linear_model)

predictions_lm = predict(linear_model,test_lm[,1:11])

Predicted_LM = data.frame(predictions_lm)

Actual = test_lm[,12]

Error = Actual - Predicted_LM

RMSE(Error)

#Error rate = 9.81 Accuracy 90.19

### ###

# PART 2 #

### ###
```

Employee Absenteeism

#Chacking Dataset

```
str(Absenteeism)
```

Creating New Dataset require to calculate Loss per month company can estimate in coming Year with same data

I.E. Loss = (Work Load * Absent Hour)/(Service Time)

```
NEW_LOSS_DF = subset(Absent, select = c(Month.of.absence, Service.time, Absenteeism.time.in.hours,  
Work.load.Average.per.day))
```

```
NEW_LOSS_DF["Loss"]=with(NEW_LOSS_DF,((NEW_LOSS_DF[,4]*NEW_LOSS_DF[,3])/NEW_LOSS_DF[,2]))
```

```
for(i in 0:12)
```

```
{
```

```
DataLOSS=NEW_LOSS_DF[which(NEW_LOSS_DF["Month.of.absence"]==i),]
```

```
print(data.frame(sum(DataLOSS$Loss)))
```

```
}
```


Appendix B – Python Code

```
# Loading Important Libraries to Read and Analyse the Data

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

%matplotlib inline

# Reading Absenteeism_at_work_Project.xls DataSet As Absenteeism & Checking Head Of the Data

Absenteeism = pd.read_excel('Absenteeism_at_work_Project.xls')

Absenteeism.head()

# Finding Missing Data and Plotting using Matplotlib library

plt.figure(figsize=(14,10))

sns.heatmap(Absent.isnull(),yticklabels=False,cbar=False,cmap='viridis')

# Calculating Missing Data value In Percentage for our Dataset Variable

Missing = pd.DataFrame(Absent.isnull().sum())

Missing = Missing.reset_index()

Missing = Missing.rename(columns={'index':'Variable',0:'Missing %'})

Missing['Missing %'] = (Missing['Missing %'] / len(Absent))*100

Missing = Missing.sort_values('Missing %',ascending=False).reset_index(drop=True)
```

Employee Absenteeism

Missing

Imputation Of Missing Data In Dataset Using Median

```
Absent['Reason for absence'] = Absent['Reason for absence'].fillna(Absent['Reason for absence'].median())
```

```
Absent['Month of absence'] = Absent['Month of absence'].fillna(Absent['Month of absence'].median())
```

```
Absent['Transportation expense'] = Absent['Transportation expense'].fillna(Absent['Transportation expense'].median())
```

```
Absent['Distance from Residence to Work'] = Absent['Distance from Residence to Work'].fillna(Absent['Distance from Residence to Work'].median())
```

```
Absent['Service time'] = Absent['Service time'].fillna(Absent['Service time'].median())
```

```
Absent['Age'] = Absent['Age'].fillna(Absent['Age'].median())
```

```
Absent['Work load Average/day'] = Absent['Work load Average/day'].fillna(Absent['Work load Average/day'].median())
```

```
Absent['Hit target'] = Absent['Hit target'].fillna(Absent['Hit target'].median())
```

```
Absent['Disciplinary failure'] = Absent['Disciplinary failure'].fillna(Absent['Disciplinary failure'].median())
```

```
Absent['Education'] = Absent['Education'].fillna(Absent['Education'].median())
```

```
Absent['Son'] = Absent['Son'].fillna(Absent['Son'].median())
```

```
Absent['Social drinker'] = Absent['Social drinker'].fillna(Absent['Social drinker'].median())
```

```
Absent['Social smoker'] = Absent['Social smoker'].fillna(Absent['Social smoker'].median())
```

```
Absent['Pet'] = Absent['Pet'].fillna(Absent['Pet'].median())
```

```
Absent['Weight'] = Absent['Weight'].fillna(Absent['Weight'].median())
```

```
Absent['Height'] = Absent['Height'].fillna(Absent['Height'].median())
```

Employee Absenteeism

Absent['Body mass index']= Absent['Body mass index'].fillna(Absent['Body mass index'].median())

Absent['Absenteeism time in hours']= Absent['Absenteeism time in hours'].fillna(Absent['Absenteeism time in hours'].median())

Absent['Reason for absence']= Absent['Reason for absence'].astype(int)

Absent['Month of absence']= Absent['Month of absence'].astype(int)

Absent['Transportation expense']= Absent['Transportation expense'].astype(int)

Absent['Distance from Residence to Work']= Absent['Distance from Residence to Work'].astype(int)

Absent['Service time']= Absent['Service time'].astype(int)

Absent['Age']= Absent['Age'].astype(int)

Absent['Work load Average/day']= Absent['Work load Average/day'].astype(int)

Absent['Hit target']= Absent['Hit target'].astype(int)

Absent['Disciplinary failure']= Absent['Disciplinary failure'].astype(int)

Absent['Education']= Absent['Education'].astype(int)

Absent['Son']= Absent['Son'].astype(int)

Absent['Age']= Absent['Age'].astype(int)

Absent['Social drinker']= Absent['Social drinker'].astype(int)

Absent['Social smoker']= Absent['Social smoker'].astype(int)

Absent['Pet']= Absent['Pet'].astype(int)

Absent['Weight']= Absent['Weight'].astype(int)

Absent['Height']= Absent['Height'].astype(int)

Employee Absenteeism

```
Absent['Body mass index'] = Absent['Body mass index'].astype(int)

Absent['Absenteeism time in hours'] = Absent['Absenteeism time in hours'].astype(int)

# Selecting all Numerical Column in one Set For Outlier Analysis

Numeric = Absent[['Transportation expense', 'Distance from Residence to Work',

                  'Service time', 'Age', 'Work load Average/day', 'Hit target', 'Son', 'Pet',

                  'Weight', 'Height', 'Body mass index', 'Absenteeism time in hours']]

# Plotting Box Plot For outlier Analysis in our Numerical Data

plt.boxplot(Numeric['Transportation expense'])

plt.boxplot(Numeric['Distance from Residence to Work'])

plt.boxplot(Numeric['Service time'])

plt.boxplot(Numeric['Age'])

plt.boxplot(Numeric['Work load Average/day'])

plt.boxplot(Numeric['Hit target'])

plt.boxplot(Numeric['Son'])

plt.boxplot(Numeric['Pet'])

# Making Set Of Numerical Columns have Outliers Present in it according to Box Plot

Outlier = Absent[['Transportation expense', 'Service time', 'Age', 'Work load Average/day', 'Hit target', 'Pet',

                  'Height', 'Absenteeism time in hours']]

for i in Outlier:
```

Employee Absenteeism

```
print(i)

q75, q25 = np.percentile(Outlier.loc[:,i], [75,25])

iqr = q75 - q25

min = q25 - (iqr*1.5)

max = q75 + (iqr*1.5)

print(min)

print(max)

#Replace with NA

#Extract quartiles

q75, q25 = np.percentile(Absent['Transportation expense'], [75,25])

#Calculate IQR

iqr = q75 - q25

#Calculate inner and outer fence

minimum = q25 - (iqr*1.5)

maximum = q75 + (iqr*1.5)

#Replace with NA

Absent.loc[Absent['Transportation expense'] < minimum,:,'Transportation expense'] = np.nan

Absent.loc[Absent['Transportation expense'] > maximum,:,'Transportation expense'] = np.nan

#Detect and replace with NA
```

Employee Absenteeism

```
#Extract quartiles
```

```
q75, q25 = np.percentile(Absent['Service time'], [75,25])
```

```
#Calculate IQR
```

```
iqr = q75 - q25
```

```
#Calculate inner and outer fence
```

```
minimum = q25 - (iqr*1.5)
```

```
maximum = q75 + (iqr*1.5)
```

```
#Replace with NA
```

```
Absent.loc[Absent['Service time'] < minimum,: 'Service time'] = np.nan
```

```
Absent.loc[Absent['Service time'] > maximum,: 'Service time'] = np.nan
```

```
#Detect and replace with NA
```

```
#Extract quartiles
```

```
q75, q25 = np.percentile(Absent['Age'], [75,25])
```

```
#Calculate IQR
```

```
iqr = q75 - q25
```

```
#Calculate inner and outer fence
```

```
minimum = q25 - (iqr*1.5)
```

```
maximum = q75 + (iqr*1.5)
```

```
#Replace with NA
```

Employee Absenteeism

```
Absent.loc[Absent['Age'] < minimum,: 'Age'] = np.nan
```

```
Absent.loc[Absent['Age'] > maximum,: 'Age'] = np.nan
```

```
#Detect and replace with NA
```

```
#Extract quartiles
```

```
q75, q25 = np.percentile(Absent['Work load Average/day'], [75, 25])
```

```
#Calculate IQR
```

```
iqr = q75 - q25
```

```
#Calculate inner and outer fence
```

```
minimum = q25 - (iqr*1.5)
```

```
maximum = q75 + (iqr*1.5)
```

```
#Replace with NA
```

```
Absent.loc[Absent['Work load Average/day'] < minimum,: 'Work load Average/day'] = np.nan
```

```
Absent.loc[Absent['Work load Average/day'] > maximum,: 'Work load Average/day'] = np.nan
```

```
#Detect and replace with NA
```

```
#Extract quartiles
```

```
q75, q25 = np.percentile(Absent['Hit target'], [75, 25])
```

```
#Calculate IQR
```

```
iqr = q75 - q25
```

```
#Calculate inner and outer fence
```

Employee Absenteeism

```
minimum = q25 - (iqr*1.5)
```

```
maximum = q75 + (iqr*1.5)
```

```
#Replace with NA
```

```
Absent.loc[Absent['Hit target'] < minimum,: 'Hit target'] = np.nan
```

```
Absent.loc[Absent['Hit target'] > maximum,: 'Hit target'] = np.nan
```

```
#Detect and replace with NA
```

```
#Extract quartiles
```

```
q75, q25 = np.percentile(Absent['Pet'], [75, 25])
```

```
#Calculate IQR
```

```
iqr = q75 - q25
```

```
#Calculate inner and outer fence
```

```
minimum = q25 - (iqr*1.5)
```

```
maximum = q75 + (iqr*1.5)
```

```
#Replace with NA
```

```
Absent.loc[Absent['Pet'] < minimum,: 'Pet'] = np.nan
```

```
Absent.loc[Absent['Pet'] > maximum,: 'Pet'] = np.nan
```

```
#Detect and replace with NA
```

```
#Extract quartiles
```

```
q75, q25 = np.percentile(Absent['Height'], [75, 25])
```


Employee Absenteeism

```
#Calculate IQR
```

```
iqr = q75 - q25
```

```
#Calculate inner and outer fence
```

```
minimum = q25 - (iqr*1.5)
```

```
maximum = q75 + (iqr*1.5)
```

```
#Replace with NA
```

```
Absent.loc[Absent['Height'] < minimum, 'Height'] = np.nan
```

```
Absent.loc[Absent['Height'] > maximum, 'Height'] = np.nan
```

```
# Imputing Median for NA obtained from Outliers
```

```
Absent['Reason for absence'] = Absent['Reason for absence'].fillna(Absent['Reason for absence'].median())
```

```
Absent['Month of absence'] = Absent['Month of absence'].fillna(Absent['Month of absence'].median())
```

```
Absent['Transportation expense'] = Absent['Transportation expense'].fillna(Absent['Transportation expense'].median())
```

```
Absent['Distance from Residence to Work'] = Absent['Distance from Residence to Work'].fillna(Absent['Distance from Residence to Work'].median())
```

```
Absent['Service time'] = Absent['Service time'].fillna(Absent['Service time'].median())
```

```
Absent['Age'] = Absent['Age'].fillna(Absent['Age'].median())
```

```
Absent['Work load Average/day'] = Absent['Work load Average/day'].fillna(Absent['Work load Average/day'].median())
```

Employee Absenteeism

Absent['Hit target']= Absent['Hit target'].fillna(Absent['Hit target'].median())

Absent['Disciplinary failure']= Absent['Disciplinary failure'].fillna(Absent['Disciplinary failure'].median())

Absent['Education']= Absent['Education'].fillna(Absent['Education'].median())

Absent['Son']= Absent['Son'].fillna(Absent['Son'].median())

Absent['Social drinker']= Absent['Social drinker'].fillna(Absent['Social drinker'].median())

Absent['Social smoker']= Absent['Social smoker'].fillna(Absent['Social smoker'].median())

Absent['Pet']= Absent['Pet'].fillna(Absent['Pet'].median())

Absent['Weight']= Absent['Weight'].fillna(Absent['Weight'].median())

Absent['Height']= Absent['Height'].fillna(Absent['Height'].median())

Absent['Body mass index']= Absent['Body mass index'].fillna(Absent['Body mass index'].median())

Absent['Absenteeism time in hours']= Absent['Absenteeism time in hours'].fillna(Absent['Absenteeism time in hours'].median())

Absent['Reason for absence']= Absent['Reason for absence'].astype(int)

Absent['Month of absence']= Absent['Month of absence'].astype(int)

Absent['Transportation expense']= Absent['Transportation expense'].astype(int)

Absent['Distance from Residence to Work']= Absent['Distance from Residence to Work'].astype(int)

Absent['Service time']= Absent['Service time'].astype(int)

Absent['Age']= Absent['Age'].astype(int)

Absent['Work load Average/day']= Absent['Work load Average/day'].astype(int)

Absent['Hit target']= Absent['Hit target'].astype(int)

Employee Absenteeism

Absent['Disciplinary failure'] = Absent['Disciplinary failure'].astype(int)

Absent['Education'] = Absent['Education'].astype(int)

Absent['Son'] = Absent['Son'].astype(int)

Absent['Age'] = Absent['Age'].astype(int)

Absent['Social drinker'] = Absent['Social drinker'].astype(int)

Absent['Social smoker'] = Absent['Social smoker'].astype(int)

Absent['Pet'] = Absent['Pet'].astype(int)

Absent['Weight'] = Absent['Weight'].astype(int)

Absent['Height'] = Absent['Height'].astype(int)

Absent['Body mass index'] = Absent['Body mass index'].astype(int)

Absent['Absenteeism time in hours'] = Absent['Absenteeism time in hours'].astype(int)

Absent['Reason for absence'] = Absent['Reason for absence'].astype('category')

Absent['Month of absence'] = Absent['Month of absence'].astype('category')

Absent['Day of the week'] = Absent['Day of the week'].astype('category')

Absent['Seasons'] = Absent['Seasons'].astype('category')

Absent['Disciplinary failure'] = Absent['Disciplinary failure'].astype('category')

Absent['Education'] = Absent['Education'].astype('category')

Absent['Social drinker'] = Absent['Social drinker'].astype('category')

Absent['Social smoker'] = Absent['Social smoker'].astype('category')

Employee Absenteeism

```
Absent['ID'] = Absenteeism['ID']
```

```
Absent['Day of the week'] = Absenteeism['Day of the week']
```

```
Absent['Seasons'] = Absenteeism['Seasons']
```

```
Missing = pd.DataFrame(Absent.isnull().sum())
```

```
Missing
```

```
# Creating Set Of Numerical Data to find Correlation Between all the data
```

```
Numerical = Absent[['ID', 'Transportation expense', 'Distance from Residence to Work',
```

```
    'Service time', 'Age', 'Work load Average/day', 'Hit target', 'Son',
```

```
    'Pet', 'Weight', 'Height', 'Body mass index']]
```

```
corr = Numerical.corr()
```

```
#Set the width and height of the plot
```

```
f, ax = plt.subplots(figsize=(10, 8))
```

```
#Plot using seaborn library
```

```
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap='rainbow', annot=True,
```

```
            square=True, ax=ax)
```

```
# Selecting ALL Feature Variable as X and Target Variable as Y for Selecting Important Feature
```

```
x = Absent[['Reason for absence', 'Month of absence', 'Day of the week',
```

```
    'Seasons', 'Transportation expense', 'Distance from Residence to Work',
```

```
    'Service time', 'Age', 'Work load Average/day', 'Hit target',
```

Employee Absenteeism

```
'Disciplinary failure', 'Son', 'Social drinker',  
  
'Pet', 'Height', 'Body mass index']]  
  
y = Absent['Absenteeism time in hours']  
  
# Performing Random Forest with all the variable to see Accuracy with 1000 Tree  
  
# Create a random forest classifier  
  
clf = RandomForestClassifier(n_estimators=1000, random_state=0, n_jobs=-1)  
  
# Train the classifier  
  
clf.fit(X_train, y_train)  
  
# Print the name and gini importance of each feature  
  
for feature in zip(feats_labels, clf.feature_importances_):  
  
    print(feature)  
  
sfm = SelectFromModel(clf, threshold=0.029)  
  
# Train the selector  
  
sfm.fit(X_train, y_train)  
  
# Selecting Important Feature  
  
for feature_list_index in sfm.get_support(indices=True):  
  
    print(feats_labels[feature_list_index])  
  
# Calculating Accuracy with All the feature to Target Variable  
  
# Applying The Full Featured Classifier To The Test Data
```

Employee Absenteeism

```
y_pred = clf.predict(X_test)

# Viewing The Accuracy Of Our Full Feature Model

accuracy_score(y_test, y_pred)

# Transforming Important Feature to important Data using SelectFromModel Package to check accuracy for important Variable

X_important_train = sfm.transform(X_train)

X_important_test = sfm.transform(X_test)

# A new random forest classifier for the most important features

clf_important = RandomForestClassifier(n_estimators=1000, random_state=0, n_jobs=-1)

# Training new classifier on the new dataset containing the most important features

clf_important.fit(X_important_train, y_train)

# Accuracy using only Important feature of Dataset

# Apply The Full Featured Classifier To The Test Data

y_important_pred = clf_important.predict(X_important_test)

# View The Accuracy Of Limited Feature Model

accuracy_score(y_test, y_important_pred)

# Checking RMSE And Accuracy for random forest and Importing Metrics library for perform mathematical metrics on our Model

from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, y_important_pred))

print('MSE:', metrics.mean_squared_error(y_test, y_important_pred))
```

Employee Absenteeism

```
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_important_pred)))

# Selecting X as our Only Important Variable and Y as Target for further Analysis for different model.

New_X = Absent[['Reason for absence', 'Month of absence', 'Day of the week', 'Work load Average/day', 'Hit target',
               'Height', 'Body mass index', 'Pet', 'Distance from Residence to Work', 'Service time', 'Age']]

New_Y = Absent['Absenteeism time in hours']

Xtrain, Xtest, Ytrain, Ytest = train_test_split(New_X, New_Y, test_size=0.3)

from sklearn.linear_model import LinearRegression

lm = LinearRegression().fit(Xtrain, Ytrain)

prediction = lm.predict(Xtest)

print('MAE:', metrics.mean_absolute_error(Ytest, prediction))

print('MSE:', metrics.mean_squared_error(Ytest, prediction))

print('RMSE:', np.sqrt(metrics.mean_squared_error(Ytest, prediction)))

from sklearn.tree import DecisionTreeRegressor

dtree = DecisionTreeRegressor()

dtree.fit(Xtrain, Ytrain)

prediction_dtree = dtree.predict(Xtest)

print('MAE:', metrics.mean_absolute_error(Ytest, prediction_dtree))

print('MSE:', metrics.mean_squared_error(Ytest, prediction_dtree))

print('RMSE:', np.sqrt(metrics.mean_squared_error(Ytest, prediction_dtree)))
```

Employee Absenteeism

```
from sklearn.ensemble import RandomForestRegressor

# Create a random forest Regressor

RFR = RandomForestRegressor(n_estimators=1000, random_state=0, n_jobs=-1)

# Train the classifier

RFR.fit(Xtrain, Ytrain)

prediction_RFR = RFR.predict(Xtest)

print('MAE:', metrics.mean_absolute_error(Ytest, prediction_RFR))

print('MSE:', metrics.mean_squared_error(Ytest, prediction_RFR))

print('RMSE:', np.sqrt(metrics.mean_squared_error(Ytest, prediction_RFR)))

# Create a random forest Regressor

RFR = RandomForestRegressor(n_estimators=2000, random_state=0, n_jobs=-1)

# Train the classifier

RFR.fit(Xtrain, Ytrain)

prediction_RFR = RFR.predict(Xtest)

# Changes Company Should Bring Using Important Feature

coeff_df = pd.DataFrame(lm.coef_, Xtest.columns, columns=['Coefficient'])

coeff_df

# Work Loss/Month
```


Employee Absenteeism

```
NEW_LOSS_DF = Absent[['Month of absence', 'Absenteeism time in hours', 'Work load Average/day', 'Service time']]
```

```
NEW_LOSS_DF["Loss"]=(NEW_LOSS_DF['Work load Average/day']*NEW_LOSS_DF['Absenteeism time in  
hours'])/NEW_LOSS_DF['Service time']
```

```
NEW_LOSS_DF["Loss"] = np.round(NEW_LOSS_DF["Loss"]).astype('int64')
```

```
Jan = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 1]['Loss'].sum()
```

```
Feb = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 2]['Loss'].sum()
```

```
Mar = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 3]['Loss'].sum()
```

```
April = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 4]['Loss'].sum()
```

```
may = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 5]['Loss'].sum()
```

```
Jun = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 6]['Loss'].sum()
```

```
Jul = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 7]['Loss'].sum()
```

```
Aug = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 8]['Loss'].sum()
```

```
Sep = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 9]['Loss'].sum()
```

```
Oct = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 10]['Loss'].sum()
```

```
Nov = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 11]['Loss'].sum()
```

```
Dec = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 12]['Loss'].sum()
```

```
data = {'Janaury': Jan, 'Febraury': Feb, 'March': Mar,
```

```
        'April': April, 'May': may, 'June': Jun, 'July': Jul,
```

```
        'August': Aug, 'September': Sep, 'October': Oct, 'November': Nov,
```

Employee Absenteeism

```
'December': Dec}
```

```
WorkLoss = pd.DataFrame.from_dict(data, orient='index')
```

```
WorkLoss = WorkLoss.rename(index=str, columns={0: "Work Load Loss/Month"})
```

```
# Work Load / Month
```

```
Jan = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 1]['Work load Average/day'].sum()
```

```
Feb = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 2]['Work load Average/day'].sum()
```

```
Mar = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 3]['Work load Average/day'].sum()
```

```
April = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 4]['Work load Average/day'].sum()
```

```
may = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 5]['Work load Average/day'].sum()
```

```
Jun = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 6]['Work load Average/day'].sum()
```

```
Jul = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 7]['Work load Average/day'].sum()
```

```
Aug = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 8]['Work load Average/day'].sum()
```

```
Sep = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 9]['Work load Average/day'].sum()
```

```
Oct = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 10]['Work load Average/day'].sum()
```

```
Nov = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 11]['Work load Average/day'].sum()
```

```
Dec = NEW_LOSS_DF[NEW_LOSS_DF['Month of absence'] == 12]['Work load Average/day'].sum()
```

```
data1 = {'Janaury': Jan, 'February': Feb, 'March': Mar,
```

```
'April': April, 'May': may, 'June': Jun, 'July': Jul,
```

```
'August': Aug, 'September': Sep, 'October': Oct, 'November': Nov,
```

Employee Absenteeism

```
'December': Dec}
```

```
WorkLossMain = pd.DataFrame.from_dict(data1, orient='index')
```

```
WorkLossMain = WorkLossMain.rename(index=str, columns={0: "Work load Average/Month"})
```

```
# Work Load Loss In Percentage % Every Month Due To Absenteeism in Company
```

```
print((WorkLoss['Work Load Loss/Month']/WorkLossMain['Work load Average/Month'])*100,"% Loss")
```