Employee Absenteeism

Vidyut Rao

5th September 2018

Table of Contents

Problem Statement

XYZ is a courier company. We know that human capital plays an important role in collection, transportation and delivery. The company is going through a genuine issue of absenteeism. We need to build a model that is capable of predicting the losses each month in the future and also bring about changes that would reduce absenteeism.

Data

Before we proceed, let us familiarize ourselves with the given data as to what the predictors are and what the target variable is.

Predictors (1-7)

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work
0	11	26.0	7.0	3	1	289.0	36.0
1	36	0.0	7.0	3	1	118.0	13.0
2	3	23.0	7.0	4	1	179.0	51.0
3	7	7.0	7.0	5	1	279.0	5.0
4	11	23.0	7.0	5	1	289.0	36.0

Predictors (8-17)

	Service time	Age	Work load Average/day	Hit target	Disciplinary failure	Education	Son	Social drinker	Social smoker	Pet
0	13.0	33.0	239554.0	97.0	0.0	1.0	2.0	1.0	0.0	1.0
1	18.0	50.0	239554.0	97.0	1.0	1.0	1.0	1.0	0.0	0.0
2	18.0	38.0	239554.0	97.0	0.0	1.0	0.0	1.0	0.0	0.0
3	14.0	39.0	239554.0	97.0	0.0	1.0	2.0	1.0	1.0	0.0
4	13.0	33.0	239554.0	97.0	0.0	1.0	2.0	1.0	0.0	1.0

Predictors (18-20) and the target variable – Absenteeism (in hours)

	Weight	Height	Body mass index	Absenteeism time in hours
0	90.0	172.0	30.0	4.0
1	98.0	178.0	31.0	0.0
2	89.0	170.0	31.0	2.0
3	68.0	168.0	24.0	4.0
4	90.0	172.0	30.0	2.0

Our dataset contains a mixed bag of binary, nominal and continuous variables. Also, it seems like the nominal data that was initially meant to be character based, has conveniently been encoded to numeric levels. For example, the *seasons* are 1, 2, 3, 4 in place of Summer, Autumn, Winter and Spring.

With the given data, we need to build a model that will predict the employee absenteeism for XYZ. We will also use multiple methods to find out which predictors greatly influence the target variable, which variable are irrelevant to the modelling process and must be dropped and in turn how to manipulate the predictors to reduce absenteeism.

Pre Processing and Exploratory Data Analysis

As the first step in any data science project, we need to explore the data for initial insights (and inconsistencies). The steps involved here include finding and imputing/eliminating missing values, analyzing outliers, analyzing the features for their strength in predicting the target variable (and eliminating multicollinearity) and scaling the features.

The primary objective of this task is to clean and smoothen the data as much as possible as better the data fed into the model, greater is the model's predictive power.

1. Preliminary Univariate Analysis

On exploring the data it was observed that a couple of columns had 0 values that do not make sense as they do not fall into any category. They were replaced with NA so that they could be imputed in the next step.

Also, we need to change the data type of the relevant categorical variables to factor (in R) or object (in Python) before we can proceed.

2. Missing Value Analysis

A first glance at the data will reveal that it contains missing values in most columns. It is common practice to drop variables that contain more than 30% missing values. The rest of the variables can have their missing values (once identified) either:

1. Deleted by dropping the records that contain them.
2. Imputing the missing values with the observation's mean or median or using the kNN (least distance) method.

If we were to drop the records in this case, we would lose a huge percentage of records. Instead, we'll go with imputation by finding out which of the three options work best.

We will begin with projecting the percentage of missing values in each section:

```
#Missing Value Analysis
miss_val = data.frame(apply(df,2,function(x){sum(is.na(x))}))
miss_val$Predictors = row.names(miss_val)
names(miss_val)[1] =  "Missing_Percentage"
miss_val$Missing_Percentage = (miss_val$Missing_Percentage/nrow(df)) * 100
miss_val = miss_val[order(-miss_val$Missing_Percentage),]
row.names(miss_val) = NULL
miss_val = miss_val[,c(2,1)]
```

Predictors	Missing_Percentage
Reason for absence	6.216216216
Body mass index	4.189189189
Absenteeism time in hours	2.972972973
Height	1.891891892
Work load Average/day	1.351351351
Education	1.351351351
Transportation expense	0.945945946
Son	0.810810811
Disciplinary failure	0.810810811
Hit target	0.810810811
Social smoker	0.540540541
Month of absence	0.540540541
Age	0.405405405
Service time	0.405405405
Distance from Residence to Work	0.405405405
Social drinker	0.405405405
Pet	0.27027027
Weight	0.135135135
Seasons	0
Day of the week	0
ID	0

None of the variables have a high percentage of missing values that warrants dropping them. We can proceed to imputation. In this case kNN imputation is intuitively the best method for imputing missing values since most fields are the same (static) based on ID and ID has no missing values. For example, for a given ID, their height, weight, BMI, Age, Service Time, number of children etc. will remain more or less the same throughout the data set. We will validate this by intentionally replacing a known observation with NA and imputing it with different methods to see which is the closest.

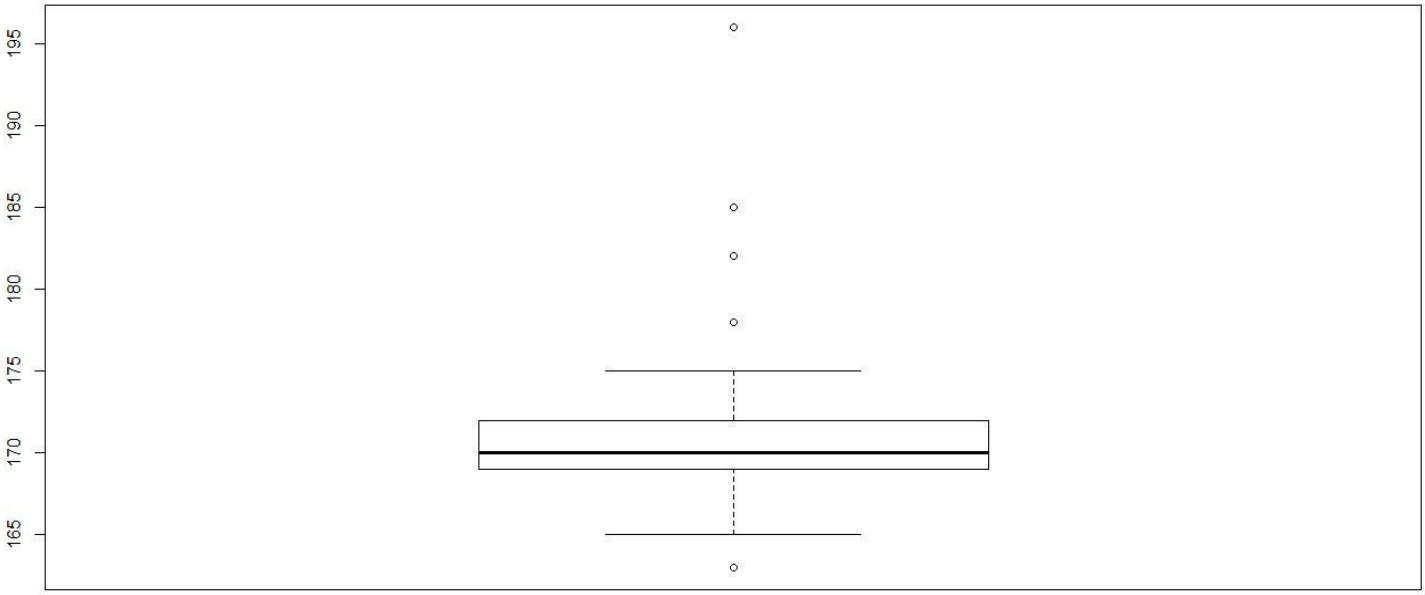
```
> df_test = df
> df_test$work.load.Average.day.[20] #
[1] 205917
> df_test$work.load.Average.day.[20] = NA
> #Test
> df_test$work.load.Average.day.[is.na(df_test$work.load.Average.day.)] =median(df_test$work.load.Avera
ge.day., na.rm = T)
> df_test$work.load.Average.day.[20] #
[1] 264249
> df_test$work.load.Average.day.[20] = NA
> df_test$work.load.Average.day.[is.na(df_test$work.load.Average.day.)] =mean(df_test$work.load.Average
.day., na.rm = T)
> df_test$work.load.Average.day.[20] #
[1] 271183.3
> df_test$work.load.Average.day.[20] = NA
> df_test = knnImputation(df_test)
> df_test$work.load.Average.day.[20] #
[1] 262552.2
```

As it can be seen, the kNN imputed value is the most accurate, so we shall freeze this method.

3. Outlier Analysis

The data needs to be inspected for extreme values that would skew and distort the data, resulting in noise that would yield a poor model. The best way to detect and remove outliers is by employing *Tukey's Boxplot Method*.

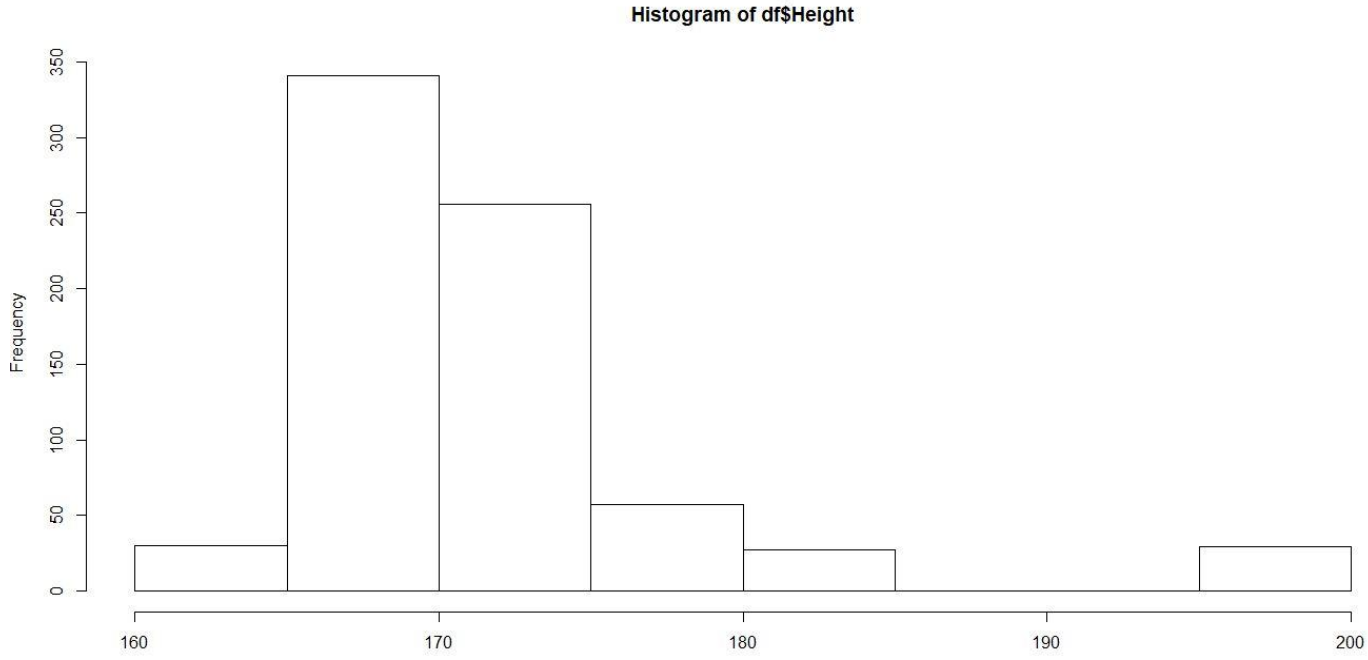
We will visualize this first with *Height* as an example:

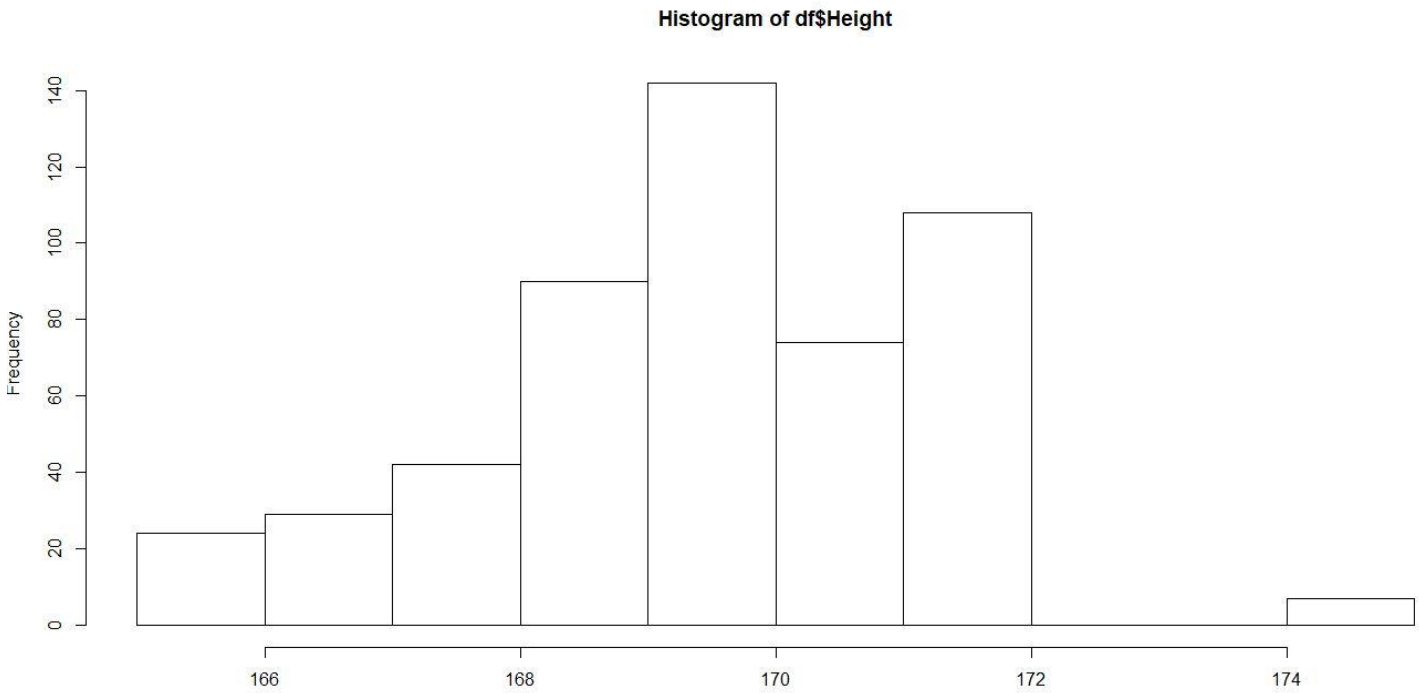


Boxplot for Height

The circles indicate the presence and location of outliers.

Below are the distributions of *Height* before and after the removal of outliers. What was previously right skewed data was corrected to a more symmetric bell curve.





I have dealt with Outliers in all variables by replacing them with NA and dropping them.

Please refer to the R and Python code files for the same since the approach is different in each case but the underlying logic of using boxplots is the same.

4. Feature Selection

Feature selection is the process of choosing a subset of the most relevant and influential features for use in our model. The main objectives of feature selection are:

- Avoid over-fitting
- Eliminating the *curse of dimensionality*
- Eliminating multicollinearity
- Simply the model and reduce the training time.

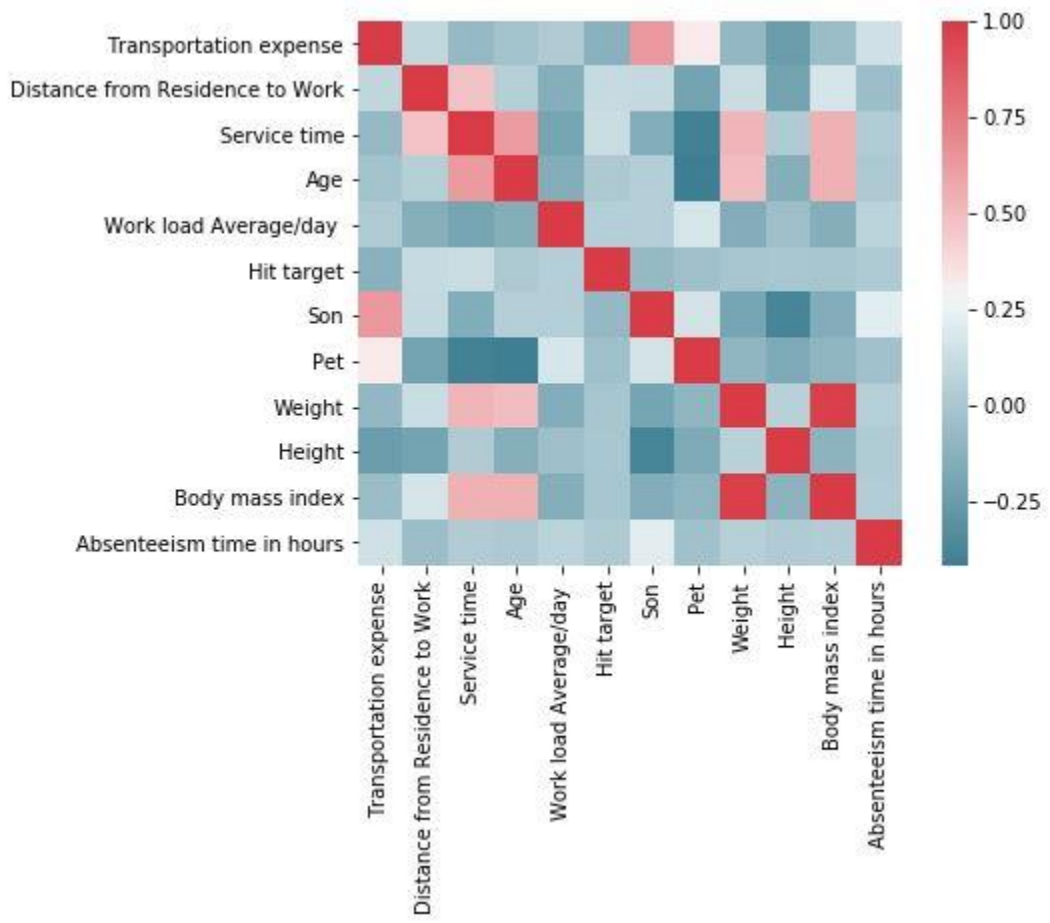
I will use Random Forests to ascertain the variable importance and create a Correlation Plot (R) or a Correlation Heat Map (Python) to find highly correlated predictors.

1. Random Forest

```
> PredImp = randomForest(Absenteeism.time.in.hours ~ ., data = df, ntree = 1000, keep.forest = FALSE, i
mportance = TRUE)
> importance(PredImp, type = 1)
                                  %IncMSE
ID                             22.9614275
Reason.for.absence             60.8705403
Month.of.absence               10.2909174
Day.of.the.week                -0.8632127
Seasons                         4.2692741
Transportation.expense         11.6933262
Distance.from.Residence.to.Work  9.0190378
Service.time                    6.8347618
Age                             9.5598958
Work.load.Average.day.          9.1603506
Hit.target                      3.9456415
Disciplinary.failure           22.7312954
Education                       0.4486586
Son                            11.0685908
Social.drinker                  6.1081031
Social.smoker                  -0.3982300
Pet                             8.4296491
Weight                          8.8464083
Height                          7.6495560
Body.mass.index                 6.6386191
```

As it can be seen, the variables *Social Smoker*, *Day of the week* and *Education* have extremely low predictive power. We will proceed to drop them.

2. Correlation



Using this heat map created with the help of the s*eaborn* library in Python, we can see that the variables *Weight* and *BMI* are very highly positively correlated. One of them will need to be dropped. Since *BMI* has greater correlation with respect to the target variable, we will retain *BMI* and drop *Weight*. The same analysis is done with the help of *symnum* in R.

```
> symnum(cor(df[,numeric_index]))
                              T D S. Ag W. H. Sn P Wg Hg B A.
Transportation.expense        1
Distance.from.Residence.to.Work  1
Service.time                   . 1
Age                              , 1
Work.load.Average.day.                1
Hit.target                               1
Son                           ,                  1
Pet                           .   .  .              1
Weight                            .  .                 1
Height                                            .       1
Body.mass.index                   .  .                 B     1
Absenteeism.time.in.hours                                        1
attr(,"legend")
[1] 0 ' ' 0.3 '.' 0.6 ',' 0.8 '+' 0.9 '*' 0.95 'B' 1
```

5. Feature Scaling

A last step in pre-processing the data is to normalize the numeric variables and standardize the range in which they fit in. The purpose of this is to ensure one variable doesn't greatly influence the model merely based on the broad range of its values. Using the logic given below, we can implement the same in R and Python

1. Normalization:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

2. Standardization:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Where σ is the standard deviation of the variable x.

Modeling

Given that we need to predict *Absenteeism* whose value is a number in hours, which makes it a continuous variable, our problem is one of ***Regression***. This was intended to be a Time Series problem, but can be achieved with ***Multiple Linear Regression*** or ***Regression Trees.***

1. Splitting the data into train and test

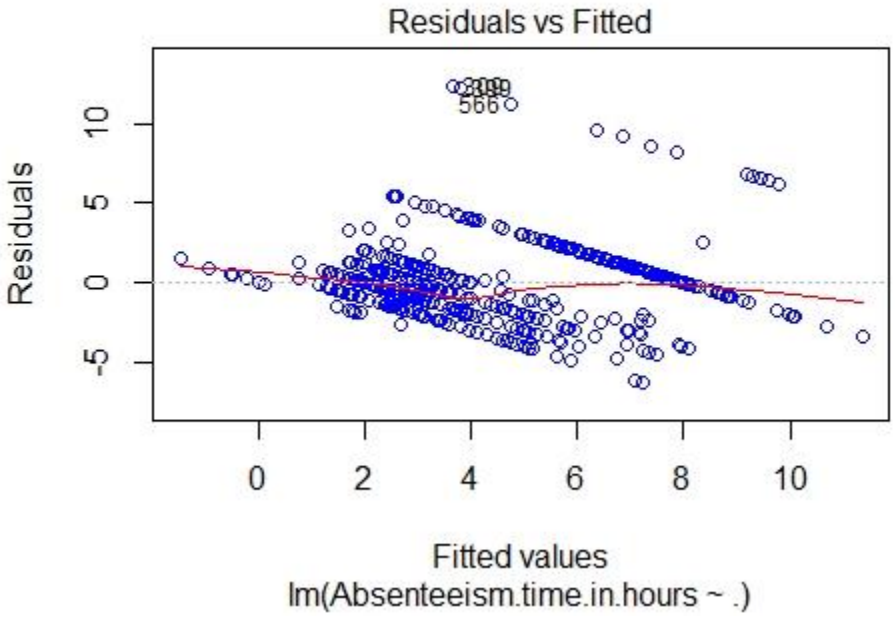
In most machine learning models, it is required that we split the data into a train set to build the model on and a test set to evaluate the model's performance against. This is not necessarily true for Regression as we can still calculate the error based on the residuals (differences between the actual values and predicted values). I have decided to split the data in Python and retain the complete data in R just to observe the impact of this decision.

2. Multiple Linear Regression

- **R**

```
> lr_model = lm(Absenteeism.time.in.hours ~., data = df)
> summary(lr_model)

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

Residuals:
    Min      1Q  Median      3Q     Max
-1.8990 -0.3929 -0.0004  0.2612  3.7582

Coefficients: (3 not defined because of singularities)
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                      0.45639    3.30322   0.138 0.890173
ID3                            -12.31274   10.66556  -1.154 0.248939
ID5                             -0.35911    5.39112  -0.067 0.946921
ID6                              4.52351    4.99386   0.906 0.365523
Seasons2                         0.23591    0.28802   0.819 0.413173
Seasons3                         0.29948    0.26354   1.136 0.256416
Seasons4                         0.03713    0.25953   0.143 0.886295
Transportation.expense          -2.20678    2.28167  -0.967 0.333979
Distance.from.Residence.to.Work       NA         NA      NA       NA
Service.time                     5.54930    8.36272   0.664 0.507305
Age                                   NA         NA      NA       NA
Work.load.Average.day.           0.07357    0.04713   1.561 0.119208
Hit.target                      -0.03413    0.05854  -0.583 0.560155
Disciplinary.failure1           -1.77747    0.28679  -6.198 1.30e-09
Son                             -1.21201    1.08435  -1.118 0.264284
Social.drinker1                       NA         NA      NA       NA
Pet                             -2.09515    2.90627  -0.721 0.471346
Body.mass.index                 -0.12669    1.10530  -0.115 0.908797
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7583 on 446 degrees of freedom
Multiple R-squared:  0.502,     Adjusted R-squared:  0.425
F-statistic: 6.516 on 69 and 446 DF,  p-value: < 2.2e-16
```

Using the *lm()* function available in the *stats* package in R, we have constructed a Linear Regression model in R. This model has an *Adjusted R-Squared* value of 0.425 which means this model explains only 42% of the data, which isn't very powerful. However, the extremely low p-value rejects the Null Hypothesis and confirms that *Absenteeism* is dependent on the predictors given.



Plot of the correlation model

- **Python**

```
#Linear Regression
lr_model = sm.OLS(train.iloc[:,16],train.iloc[:,0:16].astype('float') ).fit()

lr_pred = lr_model.predict(test.iloc[:,0:16])
```

```
lr_model.summary()
```

OLS Regression Results

Dep. Variable:	Absenteeism time in hours	**R-squared:**	0.658
Model:	OLS	**Adj. R-squared:**	0.644
Method:	Least Squares	**F-statistic:**	47.60
Date:	Tue, 04 Sep 2018	**Prob (F-statistic):**	8.51e-82
Time:	14:33:53	**Log-Likelihood:**	-1057.0
No. Observations:	412	**AIC:**	2146.
Df Residuals:	396	**BIC:**	2210.
Df Model:	16		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
ID	0.0632	0.023	2.747	0.006	0.018	0.108
Reason for absence	-0.0500	0.021	-2.420	0.016	-0.091	-0.009
Month of absence	0.1161	0.061	1.919	0.056	-0.003	0.235
Seasons	0.3369	0.170	1.978	0.049	0.002	0.672
Transportation expense	-1.0567	0.352	-3.000	0.003	-1.749	-0.364
Distance from Residence to Work	-1.8200	0.295	-6.168	0.000	-2.400	-1.240
Service time	1.1862	0.347	3.420	0.001	0.504	1.868
Age	-1.1196	0.289	-3.873	0.000	-1.688	-0.551
Work load Average/day	0.2883	0.178	1.617	0.107	-0.062	0.639
Hit target	0.3903	0.193	2.022	0.044	0.011	0.770
Disciplinary failure	-4.0803	0.947	-4.309	0.000	-5.942	-2.219
Son	1.1261	0.233	4.842	0.000	0.669	1.583
Social drinker	4.8730	0.781	6.236	0.000	3.337	6.409
Pet	0.2431	0.267	0.912	0.362	-0.281	0.767
Height	-0.3023	0.206	-1.470	0.142	-0.707	0.102
Body mass index	0.0123	0.255	0.048	0.962	-0.490	0.514

Omnibus:	111.054	**Durbin-Watson:**	2.012
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	278.451
Skew:	1.323	**Prob(JB):**	3.43e-61
Kurtosis:	6.037	**Cond. No.**	180.

The model in Python looks more promising since it has a much higher *Adjusted R-Squared* value of 0.644.

3. Regression Trees

In addition to Multiple Linear Regression, we can also use ***decision trees*** or ***random forests*** to build regression models to predict our dependent variable. Here we have used the ***rpart()*** function available in the *rpart* library in R and ***RandomForestRegressor()*** available *in sklearn.ensemble* in Python. The performance of these models have been evaluated in the subsequent section.

Model Evaluation

Error Metrics

Now that we have trained a few models, it is time we evaluate their predictive power. In the case of Regression, we have multiple approaches. Predictive performance can be measured by comparing predictions of the models with real values of the target variables, and calculating some average error measure. In case of Python, we have test data to predict and evaluate the predictions against. We use ***MAE*** and ***MSE*** to identify the errors involved.

1. Mean Absolute Error:
   This is calculated by taking the mean of the absolute differences between the actual value and its corresponding prediction.
2. Mean Squared Error:
   This is calculated by taking the mean of the squares of the differences between the actual value and its corresponding prediction.

```
#Error Metrics
mae = function(y,yhat){
  mean(abs(y-yhat))
}

mse = function(y,yhat) {
  mean((y-yhat)^2)
}
```

Evaluating the models

- **R**

```
> mae(df[,17],pred_lr)
[1] 1.580922
> mse(df[,17],pred_lr)
[1] 5.372164
> mae(df[,17],pred_rt)
[1] 1.613095
> mse(df[,17],pred_rt)
[1] 5.478503
```

- **Python**

```
#Linear Regression Error Metrics
print(MAE(test.iloc[:,16], lr_pred))
print(MSE(test.iloc[:,16], lr_pred))
```

```
2.528648593806256
10.909555309737097
```

```
#Random Forest Error Metrics
print(MAE(test.iloc[:,16], rt_pred))
print(MSE(test.iloc[:,16], rt_pred))
```

```
1.8377982879534929
7.786730169165421
```

Overall, we seem to be getting pretty consistent results in either of the models used, in both R and Python. In Python, Random Forest Regression seems to be slightly more accurate but not enough to make a decision.

Hence, we can choose either model without much loss in information.

Closing Thoughts

We have built a model to predict the *Absenteeism* in the future. Please refer to the ***Absenteeism_R_Code.R*** and ***Absenteeism_Python.ipynb*** files for the code. This will help XYZ estimate the losses they would incur if the same trend of absenteeism continues.

Based on the variable importance we have seen earlier, ***Reason for Absence***, ***ID*** and ***Disciplinary Failure*** seem to the most influential in predicting *Absenteeism*. With this, the following changes may be implemented:

1. Seeing how we can't influence the exact reason for being absent, many hours are lost in *medical* or *dental consultation*. Perhaps having a consultant visit XYZ for monthly check-ups will reduce the absenteeism for these reasons.
2. Those IDs with high rates of *Absenteeism* and *Disciplinary Failures* can be let go and hire new resources in their place.
   (For example ID 36 and 28)