Employee Absenteeism

Gideon Anthony D’Souza

Jan 2019

Contents

Chapter 1

Introduction

1.1 Problem Statement

The objective of this Case is as follows.

- What changes company should bring to reduce the number of absenteeism?
- How much losses every month can we project in 2011 if same trend of absenteeism continues?

1.2 Data

In order to predict future absenteeism based on the given data, it is required to build a regressor model on the data set provided. Table 1.1, 1.2, Table 1.3 & 1.6 below is a sample of the data set that is using to predict employee absenteeism.

Table 1.1: Absenteeism (Columns(1-6))

ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense
11	26.0	7.0	3	1	289.0
36	0.0	7.0	3	1	118.0
3	23.0	7.0	4	1	179.0
7	7.0 7.0	5	1	1	279.0
11	23.0	7.0	5	1	289.0

Table 1.2: Absenteeism (Columns(7-11)

Distance from Residence to Work	Service time	Age	Work load Average/day	Hit target
36.0	13.0	33.0	239554.0	97.0
13.0	18.0	50.0	239554.0	97.0
51.0	18.0	38.0	239554.0	97.0
5.0	14.0	39.0	239554.0	97.0
36.0	13.0	33.0	239554.0	97.0

Table 1.3: Absenteeism (Columns(12-17)

Disciplinary failure	Education	Son	Social drinker	Social smoker	Pet
0.0	1.0	2.0	1.0	0.0	1.0
1.0	1.0	1.0	1.0	0.0	0.0
0.0	1.0	0.0	1.0	0.0	0.0
0.0	1.0	2.0	1.0	1.0	0.0
0.0	1.0	2.0	1.0	0.0	1.0

Table 1.4: Absenteeism (Columns(18-22)

Weight	Education	Height	Body mass index	Absenteeism time in hours
90.0	172.0	30.0	4.0	
98.0	178.0	31.0	0.0	
89.0	170.0	31.0	2.0	
68.0	168.0	24.0	4.0	
90.0	172.0	30.0	2.0	

1.2.1 Data Overview:

As shown in the Table1.7 there are 21 features or predictor variables to predict the employee absenteeism which is the target or label. There is one target variables as shown in Table1.7.

Table 1.5:

S.No	Predictor	Target
1	ID	Absenteeism time in hours
2	Reason for absence	
3	Month of absence	
4	Day of the week	
5	Seasons	
6	Transportation expense	
7	Distance from Residence to Work	
8	Service time	
9	Age	
10	Work load Average/day	
11	Hit target	
12	Disciplinary failure	
13	Education	
14	Son	
15	Social drinker	
16	Social smoker	
17	Pet	
18	Weight	
19	Education	
20	Height	
21	Body mass index	

1.2.2 Data Description

From the given in the metadata file, the feature and target variables names and descriptions are given in table 1.6

Table 1.6: Feature and target description. Detailed description for features with **

S.No	Predictor/target variables	Description
1	ID	Employee ID
2	Reason for absence	Reason for absence**
3	Season	Season (1:spring, 2:summer, 3:fall, 4:winter)
4	Transportation expense	travel expense
5	Month of absence	Month (1 to 12)
6	Distance from Residence to Work	Distance in km
7	Day of the week	Monday - Friday (2-5)
8	Service time	Years employed in company.
9	Age	Age of employee
10	Work load Average/day	work load per day
11	Hit target	target achieved in percentage
12	Disciplinary failure	Disciplinary failure(0-no 1-yes)
13	Education	Education.**
14	Son	number of sons.
15	Pet	number of pets.
16	Social drinker	0 - no, 1 - yes.
17	Social smoker	0 - no, 1 - yes.
18	Weight	employee in kg.
19	Height	Height of employee.
20	Body mass index	employee BMI.
21	Absenteeism time in hours	Absenteeism in hours

- **Reason for absence:**
  * Certain infectious and parasitic diseases
  * Neoplasms
  * Diseases of the blood and blood-forming organs and certain disorders involving the immune mechanism
  * Endocrine, nutritional and metabolic diseases
  * Mental and behavioral disorders
  * Diseases of the nervous system
  * Diseases of the eye and adnexaVIII Diseases of the ear and mastoid process
  * Diseases of the circulatory system
  * Diseases of the respiratory system
  * Diseases of the digestive system
  * Diseases of the skin and subcutaneous tissue
  * Diseases of the musculoskeletal system and connective tissue
  * Diseases of the genitourinary system

* Pregnancy, childbirth and the puerperium
* Certain conditions originating in the perinatal period
* Congenital malformations, deformations and chromosomal abnormalities
* Symptoms, signs and abnormal clinical and laboratory findings, not elsewhere classified
* Injury, poisoning and certain other consequences of external causes
* External causes of morbidity and mortality
* Factors influencing health status and contact with health services.
* medical consultation
* blood donation
* laboratory examination
* unjustified absence
* physiotherapy
* dental consultation.

– **Education:**
high school (1), graduate (2), postgraduate (3), master and doctor (4).

From the information provided by the metadata. The continuous and categorical feature variables can clearly be separated. Table 1.7 shows the categorical and continuous variables provided in the data set.

Table 1.7: List of categorical ans continuous features present in the data set.

S.No	Categorical features	Continuous features
1	Season	Transportation expense
2	ID	Distance from Residence to Work
3	Reason for absence	Service time
4	Month of absence	Age
5	Day of the week	Work load Average/day
6	Disciplinary failure	Hit target
7	Education	Weight
8	Son	Height
9	Pet	Body mass index
10	Social drinker	Absenteeism time in hours
11	Social smoker	

Chapter 2

Data Pre-Processing

2.1 Data Preparation

In order to create a model for the data it is first required to look into the data. First the data types of the feature and target variables are checked in order to see if it complies with the information given in the meta data file (data description file). In order to perform the aforementioned the variables are split according to their data types by the function listing **??**.

Listing 2.1: Function to seperate the variables names into their respective data types.

```
def dtype_separator(df, col_names):
    '''
    ###################################################################
    This function will segregate the different data types present in the dataframe.
    ##################################################################
    Input -
    * df - The data frame to be analysed.
    *A list of the column names.
    Output -
    * obj - columns containing object data type.
    * num - columns containing numerical data type (int64/float64).
    * bool_d - columns containing bool data type.
    * unknown = columns containing data type such as datetime64, timedelta[ns] and others.
    ##################################################################
    '''
    obj = []
    num = []
    bool_d = []
    unknown = []
    for i in range(len(col_names)):
```

```
        if df.iloc[:,i].dtype == 'O':
            obj.append(col_names[i])
        elif df.iloc[:,i].dtype == 'int64' or df.iloc[:,i].dtype == '↩
           float64':
            num.append(col_names[i])
        elif df.iloc[:,i].dtype == 'bool':
            bool_d.append(col_names[i])
        else:
            unknown.append(col_names[i])
    return obj, num, bool_d, unknown
```

Function **??** outputs list of columns according to their data types. It was noted that the categorical features were incorrectly saved as numerical data type and hence they are to be converted back to categorical data type. It was also noted that the months feature ranges from 0-12, hence the value 0 is converted to NA as there is only 12 months in a year. Work load average is divided by 1000 as it was informed by the support team i a ticket raised by me.

2.2 Missing Value Analysis

The data set was checked for missing values and was found that Body mass index has the highest percentage of missing value of 4%. The missing value percentage for each feature is shown in figure 2.1. In order to deal with the missing values we impute them using KNN with a k value of 3. This would compute the k nearest values and replace the na with the respective k nearest value.

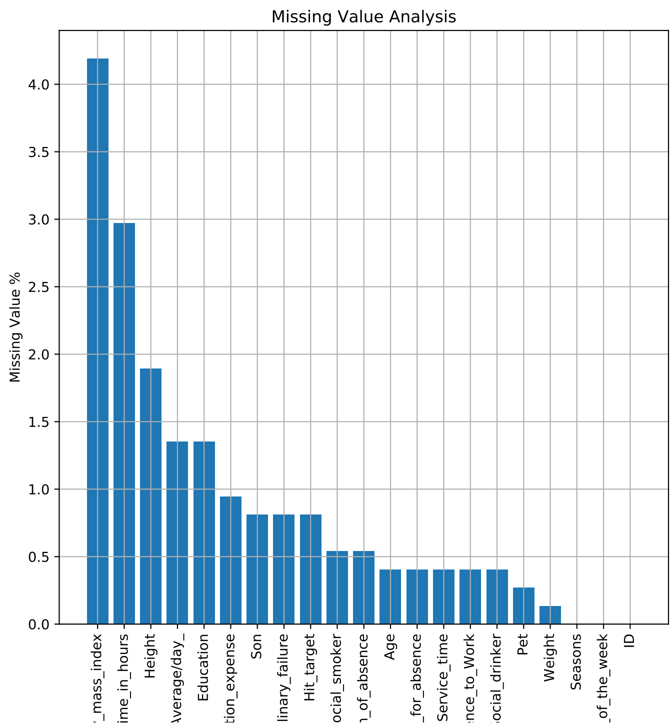


Figure 2.1: Missing value percentage for each feature.

After conducting the missing value analysis, the distributions of the continuous features are plotted. This is shown in figure 2.2, 2.3, 2.4, 2.5 & 2.6. Figure 2.7 is the distribution plot of the target variables. It can be noted that the data is not nominally distributed and after analysis the skewness it can be seen that there is a slight skew as shown in figure 2.8. We will conduct an outlier analysis and see if this affects the distributions.

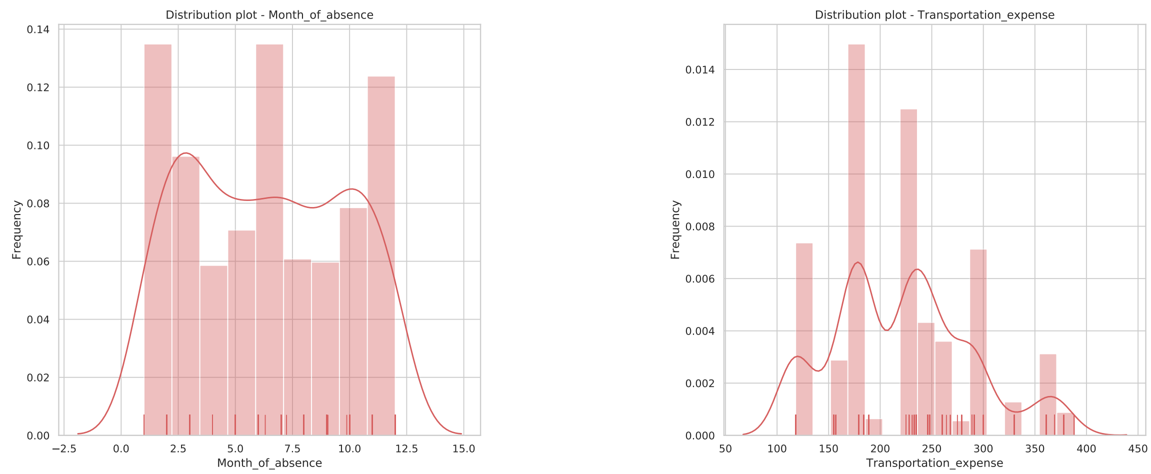


Figure 2.2: Distribution plots of features before outlier analysis.

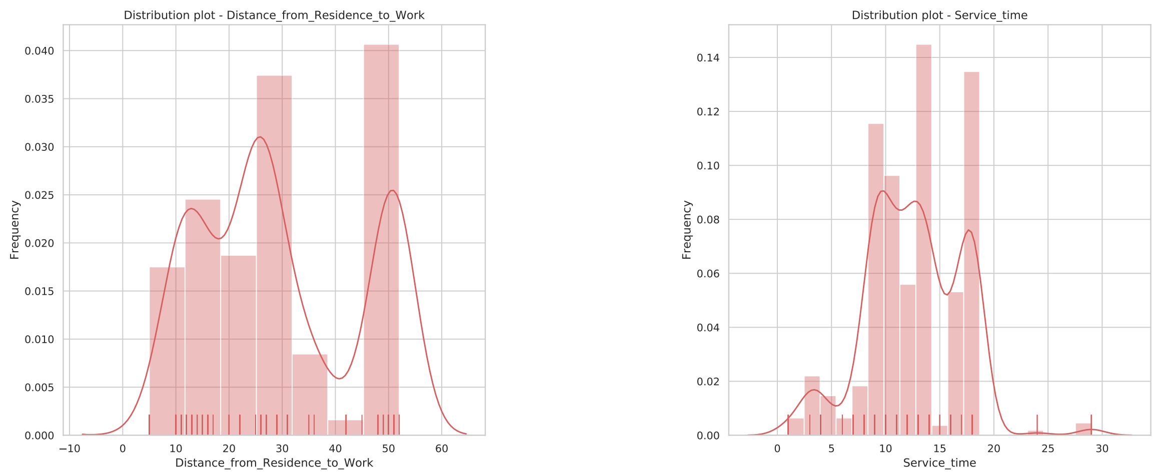


Figure 2.3: Distribution plots of features before outlier analysis.

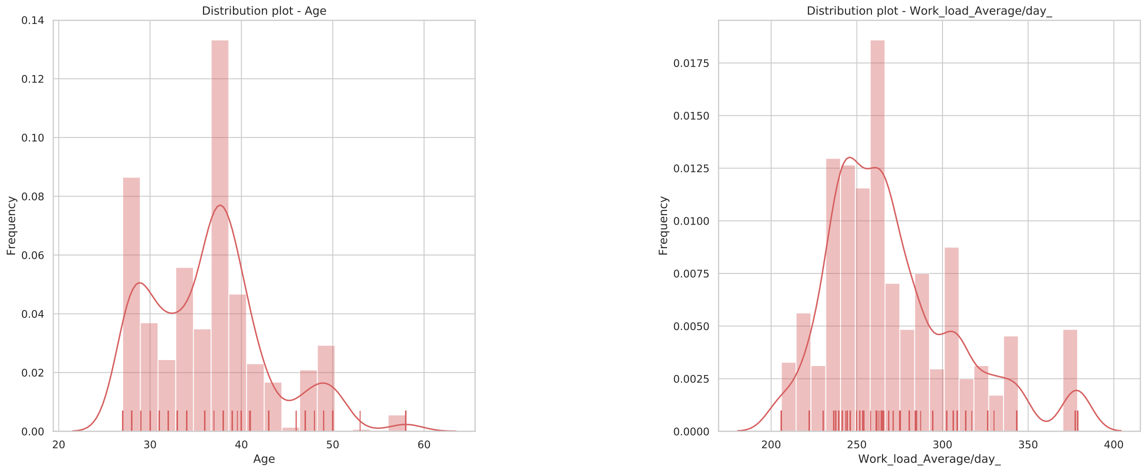


Figure 2.4: Distribution plots of features before outlier analysis.

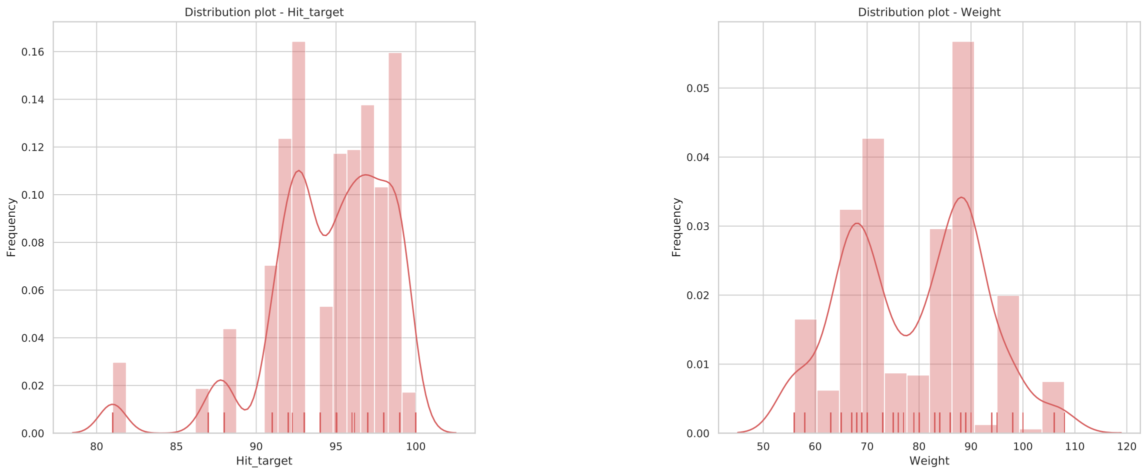


Figure 2.5: Distribution plots of features before outlier analysis.

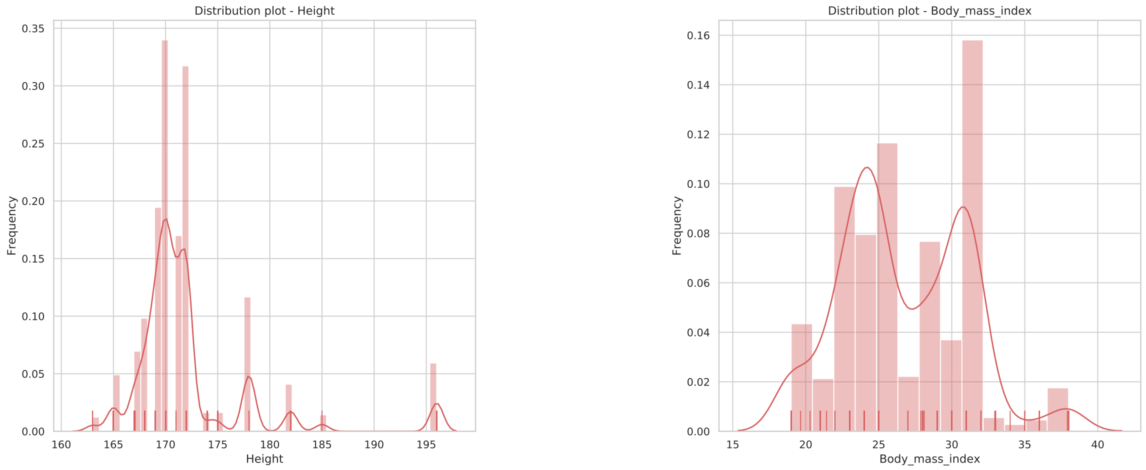


Figure 2.6: Distribution plots of features before outlier analysis.

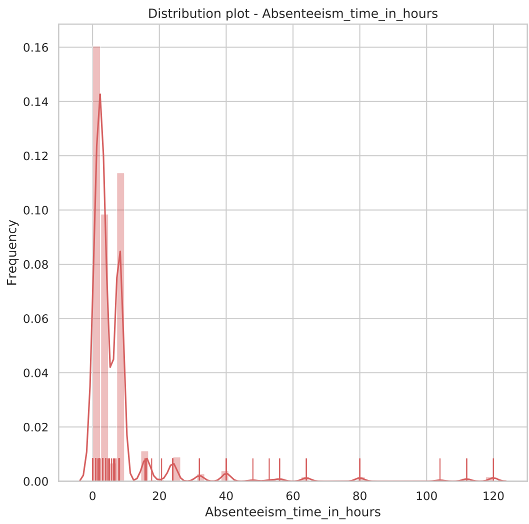


Figure 2.7: Distribution plots of target before outlier analysis.

```
*******************************************************************
statistical properities of Month_of_absence:
DescribeResult(nobs=740, minmax=(1.0, 12.0), mean=6.354700578338823, variance=11.65477971915844, skewness=0.070265400731
89858, kurtosis=-1.2589112442736288)
*******************************************************************
statistical properities of Transportation_expense:
DescribeResult(nobs=740, minmax=(118.0, 388.0), mean=221.31365847071507, variance=4480.378835673238, skewness=0.39566180
701090514, kurtosis=-0.3221653522364045)
*******************************************************************
statistical properities of Distance_from_Residence_to_Work:
DescribeResult(nobs=740, minmax=(5.0, 52.0), mean=29.631081080006933, variance=220.1302910973327, skewness=0.31144982841
84064, kurtosis=-1.2612709700406315)
*******************************************************************
statistical properities of Service_time:
DescribeResult(nobs=740, minmax=(1.0, 29.0), mean=12.554054054054054, variance=19.22711480086311, skewness=-0.0047099912
53832424, kurtosis=0.6704067197065524)
*******************************************************************
statistical properities of Age  :
DescribeResult(nobs=740, minmax=(27.0, 58.0), mean=36.44789531460408, variance=41.95812343690743, skewness=0.69731597063
86888, kurtosis=0.42388967427125657)
*******************************************************************
statistical properities of Work_load_Average/day_:
DescribeResult(nobs=740, minmax=(205.917, 378.884), mean=271.1603506483232, variance=1507.4032516337088, skewness=0.9711
718700009127, kurtosis=0.6453444214427804)
*******************************************************************
statistical properities of Hit_target:
DescribeResult(nobs=740, minmax=(81.0, 100.0), mean=94.58582517619392, variance=14.28251421473298, skewness=-1.258105085
860628, kurtosis=2.39243588672608)
*******************************************************************
statistical properities of Weight:
DescribeResult(nobs=740, minmax=(56.0, 108.0), mean=79.03513513491443, variance=165.9771129815394, skewness=0.0169668903
97851576, kurtosis=-0.9158612722683972)
*******************************************************************
statistical properities of Height:
DescribeResult(nobs=740, minmax=(163.0, 196.0), mean=172.11497717710074, variance=36.42001797662386, skewness=2.56095929
61306387, kurtosis=7.260198747582171)
*******************************************************************
statistical properities of Body_mass_index:
DescribeResult(nobs=740, minmax=(19.0, 38.0), mean=26.6813915575566, variance=18.366170391434352, skewness=0.29994337932
45435, kurtosis=-0.32299568705714377)
*******************************************************************
statistical properities of Absenteeism_time_in_hours:
DescribeResult(nobs=740, minmax=(0.0, 120.0), mean=7.018145823809551, variance=179.92603279103142, skewness=5.6323240010
94293, kurtosis=37.50800276377008)
*******************************************************************
```

Figure 2.8: Skwness test before outlier analysis.

2.3 Outlier Analysis

From figure2.2, 2.3, 2.4, 2.5, 2.6, 2.7 & **??** it can be observed that most of the variables are skewed. This could be due to outliers present in the data set. In order to visualize outliers, the classic approach of using boxplot is used. Figure 2.9 & 2.10 Shows the outlier analysis using boxplot. It can be seen from figure 2.9there that hight variables have outliers present in them and figure 2.10 shows the presence of outliers in the target variable. **Note: Outlier where also present in other features and their output is attached in the appendix.** In order to decide as to how to deal with the outliers in the data set, the percentage of outliers in the data set is first calculated by sample code in listing 2.2. the output of listing 2.2 shows that the **percentage of outliers present in the data set in 31%**. In such a case it was decided to use knn imputation to compute the values of the outliers and replace them with it.

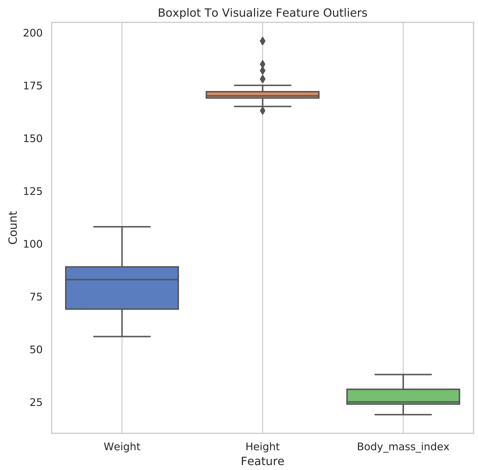


Figure 2.9: Outliers present in the feature variables.

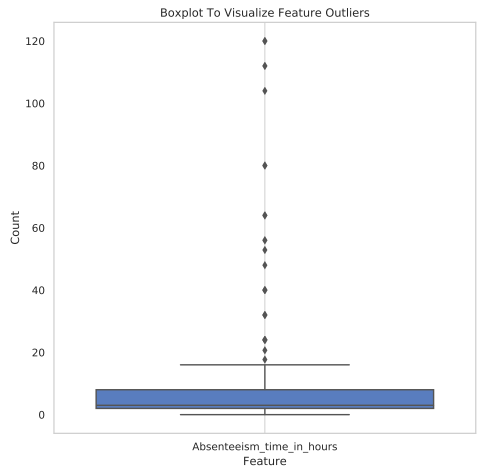


Figure 2.10: Outliers present in the target variables.

Listing 2.2: Function to calculate outlier percentage.

```
l_q = df.quantile(0.25)
u_q = df.quantile(0.75)
iqr = u_q - l_q
total_outliers = ((df<(l_q - 1.5*iqr))|(df>(u_q + 1.5*iqr))).sum()
outliers = total_outliers.sum()

r,c = df.shape
outlier_percentage = (outliers/r)*100
print("Outlier percentage :",round(outlier_percentage,3))
```

Now that the outliers are replaced with their respective knn results, the distributions of the continuous variables are compared. For the purpose of the report the distribution for absenteeism time in hours is analyzed. The rest of the distributions can be seen in the appendix section. From figure **??** it can be seen that after outlier analysis, the maximum value of the distribution is reduced from 120 to 16. It can also be noted that the shape of the distribution is changed. The distribution still does not resemble a normal distribution. The distribution seems more likely to be Poisson in nature.

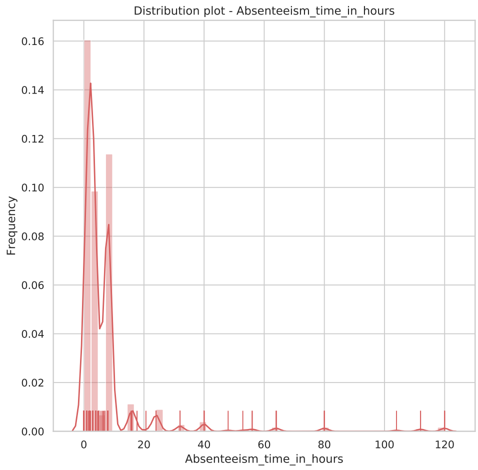


Figure 2.11: Distribution of the target variable before outlier analysis.

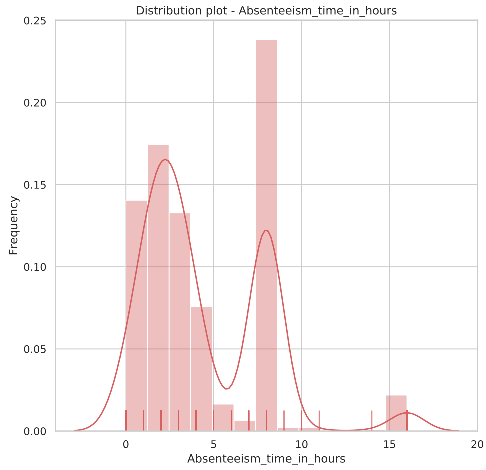


Figure 2.12: Distribution of the target variable before outlier analysis.

2.4 Data Understanding

In order to get further insight and understand of the data set. It required is to see as to how different features interact with each other and the target. First the amount of absenteeism counts verses employee id is analyzed. this is shown in figure 2.13. It can be seen that employee 9 and 28 has the maximum absenteeism count for between the time 0-5 hours and the reason is for medical consultation. it is also noted that employee id 17, 11 and 24 takes the maximum amount of unjustified absence and are mainly absent for the whole day that is 10 hours.

Next we shall analyses the education and absenteeism count along side the reason. From figure 2.4 it can be said that employees with only a high school degree tend to be absent the most and the most often used reason is medical consultation. Where as other employees with higher qualification are absent significantly less.

Now we check as to how absenteeism varies from pet owners and people who have children. From figure 2.4 By plotting number of children verses absentees it can be seen that employees with no children tend to be absent more often that the others. Next by plotting number of pets verses absentees it can be seen that employees with no pets tend to be absent more often that the others.

Next we analysis the month of absence with absenteeism count. From 2.18 we can see that the employee absentees is almost the same through the year.

Next we analysis the work load per day vs absenteeism count with respect to service time and age. From 2.19 we can see that the employee having medium amount of work load i.e.240 - 300 work load per day and who are below the age of 20 and having service time less than 8 years tend to be absent more often.

We now analysis the absenteeism vs month of absenteeism with respect to education. From 2.20 it is clearly seen that employees having only a high school tend to be absent more often than the other employees.

Figure 2.13: Analysis between employee id and absenteeism count. It can be seen that employee 9 and 28 has the maximum absenteeism count for between the time 0-5 hours and the reason is for medical consultation. it is also noted that employee id 17, 11 and 24 takes the maximum amount of unjustified absence and are mainly absent for the whole day that is 10 hours. The categories are as follows 1 - Code of Diseases, 2 - medical consultation, 3 - laboratory consultation, 4 - unjustified absence and 5 - physiotherapy

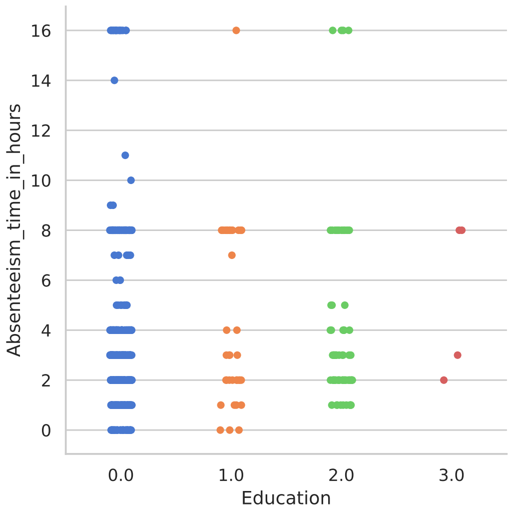


Figure 2.14: By plotting education qualification verses absenteesm it can be seen that employees with only a high school qualification tend to be absent more often that the others. Education (high school (1), graduate (2), postgraduate (3), master and doctor (4))

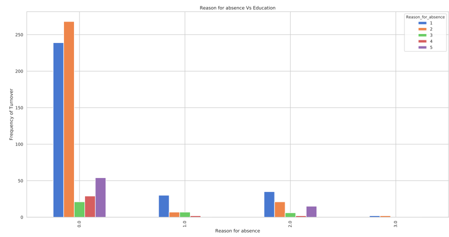


Figure 2.15: By plotting education qualification verses absentees with respect to the reason of absenteeism. it can be said that employees with only a high school degree tend to be absent the most and the most often used reason is medical consultation. Where as other employees with higher qualification are absent significantly less.

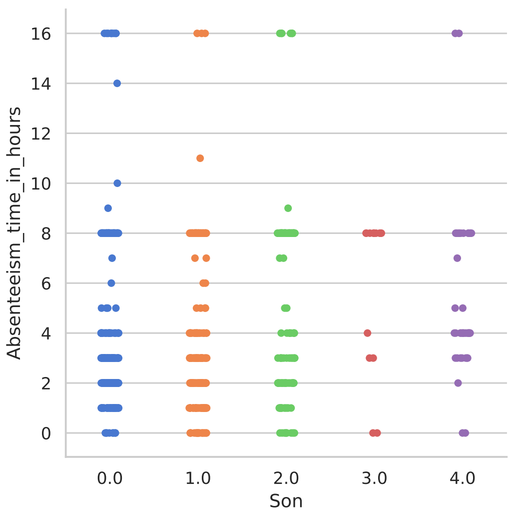


Figure 2.16: By plotting number of children verses absentees it can be seen that employees with no children tend to be absent more often that the others.

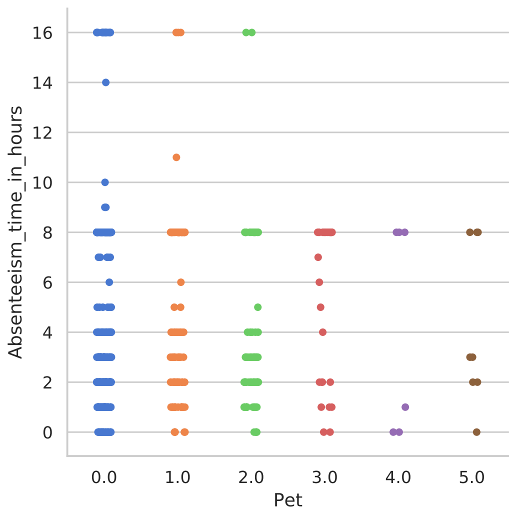


Figure 2.17: By plotting number of pets verses absentees it can be seen that employees with no pets tend to be absent more often that the others.

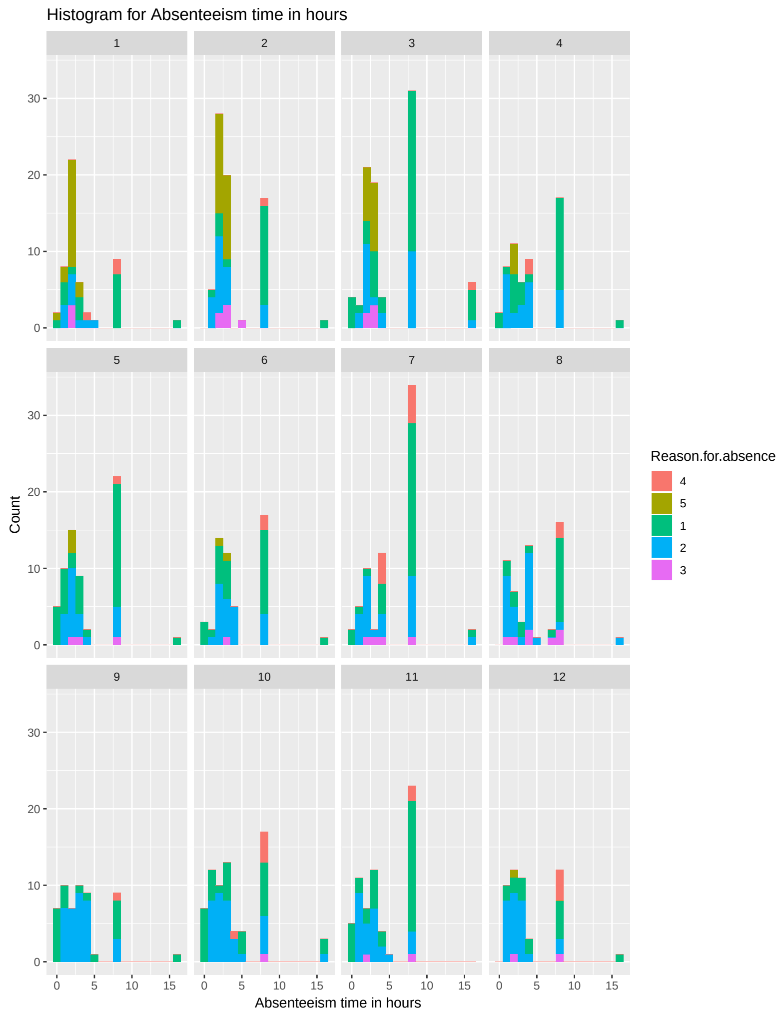


Figure 2.18: By plotting the month of absence verses reason. we can see that the employee absentees is almost the same through the year.

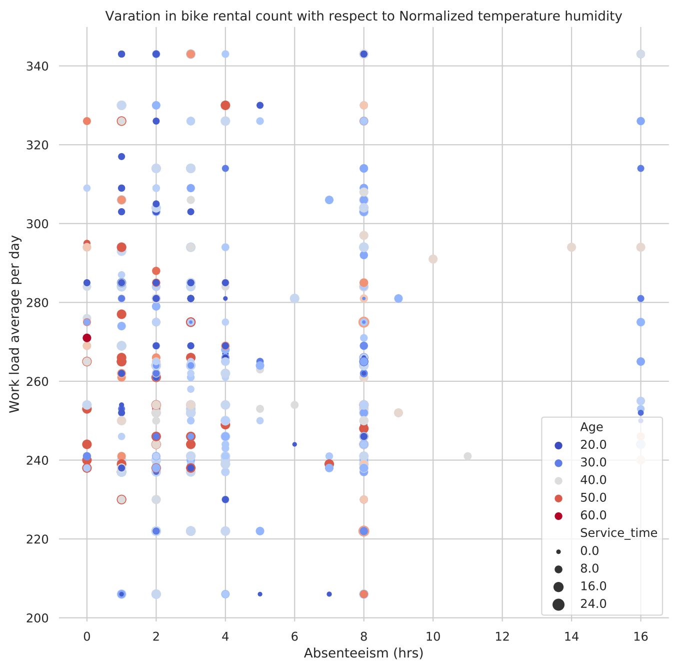


Figure 2.19: By plotting the work load per day vs absenteeism count with respect to service time and age. We can see that the employee having medium amount of work load i.e.240 - 300 work load per day and who are below the age of 20 and having service time less than 8 years tend to be absent more often.

Figure 2.20: By plotting the absenteeism vs month of absenteeism with respect to education. It is clearly seen that employees having only a high school tend to be absent more often than the other employees.

Chapter 3

Feature Selection

To develop an efficient model, it is required to know which are the features which are of most importance in predicting the target variable. It is possible that many variable in our data set is less important in predicting the target than others or they my just be redundant. In this case it is required that we remove these variables from our data set to reduce its complexity. At times two features may carry the same information. In such case it is required to remove one of the features to avoid multi collinerity problems. There are many way to perform feature selection or dimensionality reduction, but the method selected is this report is **Correlation Analysis** for continuous features and **ANOVA test** for categorical features.

3.0.1 Correlation Analysis

Correlation Analysis is a technique which helps to determine how strongly two features are related to each other(i.e their co-variance). As the co-variance can vary from - infinity to + infinity, the correlation is used as it is a scaled version of the co-variance having values ranging from -1 to +1. A correlation plot is shown in figure 3.1. A correlation threshold of 0.8 is set and feature pairs of which exceeds this threshold, one of them is dropped. By examining table figure 3.1 it can be observed that features body mass index and weight are highly correlated. Hence the feature body mass index is dropped from the analysis.

3.0.2 Analysis Of Variance (ANOVA)

Analysis of Variance (ANOVA) is a statistical technique used to compute and compare the mean between two or more groups of observations. ANOVA makes use of two variables which are categorical variables and numeric variables of the data set. The python code in listing 3.1 is used to compute the p values of the feature and target variables. listing 3.1 also the output p- values to the threshold value of 0.05 and saves the feature names to be dropped in drop_feat variable. The output of the code in listing 3.1 is shown in figure 3.2.

Listing 3.1: The python code used to comput the p values of the feature and target variables

```
## ANOVA TEST FOR P VALUES
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

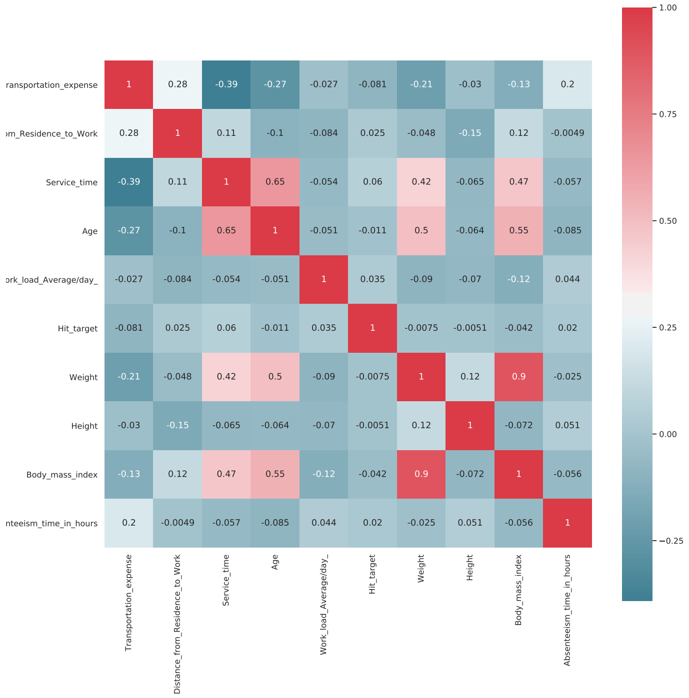


Figure 3.1: Correlation analysis of continuous variables.

```

anova_p = []
for  i in obj_dtype:
    buf = label + ' ~ ' + i
    mod = ols(buf,data=df).fit()
    anova_op = sm.stats.anova_lm(mod, typ=2)
    print(anova_op)
    anova_p.append(anova_op.iloc[0:1,3:4])
    p = anova_op.loc[i,'PR(>F)']
    if p >= 0.05:
        drop_feat.append(i)
```

From figure 3.2 we can see that features Day_of_the_week, Season, Social_smoker and Education are greater than the p value threshold (0.05) and hence they are dropped along with Body_mass_index. Now the number of variables have been reduced from 21 to 16.

```
             sum_sq     df         F        PR(>F)
ID        1489.036365   35.0  4.146099  1.148030e-13
Residual  7223.875798  704.0       NaN           NaN
                      sum_sq     df          F        PR(>F)
Reason_for_absence  1397.477786    4.0  35.102023  7.515585e-27
Residual            7315.434376  735.0        NaN           NaN
                     sum_sq     df         F    PR(>F)
Month_of_absence   282.115342   11.0  2.214608  0.01228
Residual          8430.796821  728.0       NaN      NaN
                    sum_sq     df         F    PR(>F)
Day_of_the_week   67.201829    4.0  1.428262  0.222809
Residual        8645.710333  735.0       NaN       NaN
             sum_sq     df         F    PR(>F)
Seasons     69.029361    3.0  1.959213  0.118713
Residual  8643.882801  736.0       NaN       NaN
                         sum_sq     df          F        PR(>F)
Disciplinary_failure   581.734796    1.0  52.799276  9.418497e-13
Residual              8131.177366  738.0        NaN           NaN
              sum_sq     df         F    PR(>F)
Education   35.559279    3.0  1.005362  0.389784
Residual  8677.352883  736.0       NaN       NaN
             sum_sq     df        F    PR(>F)
Son       339.960097    4.0  7.46065  0.000007
Residual  8372.952065  735.0      NaN       NaN
                   sum_sq     df         F    PR(>F)
Social_drinker   73.090287    1.0  6.243257  0.012683
Residual       8639.821875  738.0       NaN       NaN
                  sum_sq     df         F    PR(>F)
Social_smoker   23.110899    1.0  1.962743  0.161641
Residual      8689.801263  738.0       NaN       NaN
             sum_sq     df         F    PR(>F)
Pet       150.756818    5.0  2.584758  0.024951
Residual  8562.155344  734.0       NaN       NaN
```

Figure 3.2: Analysis Of Variance (ANOVA).

Chapter 4

Feature Engineering

It was noticed that the categories 1-21 in the feature Reasons_for_absence fall under one common theme which is Code of Diseases. Hence categories 1-21 are combined as a single category in the feature Reason_for_absence. In the same feature categories 22,23 and 28 relate to some kind of medical consultation and categories 24 and 25 relate to some kind of laboratory visit. Hence these categories are combined to form a single category for each respectively. Now the number of categories present in Reason_for_absence has reduced from 28 to 4.

4.0.1 Feature Scaling:

After the outlier analysis and other pre processing another skewness test is conducted on the data to check if they are skewed. As show in figure 4.1 Absenteeism_time_in_hours has a skew of about 1.1 which is much greater than rest of the features. In order to reduce the skewness of Absenteeism_time_in_hours a log transform is performed on Absenteeism_time_in_hours. This reduces the skewness of Absenteeism_time_in_hours to -0.23 as shown in figure 4.2.

After this the rest of the data is standardized in order to avoid any bias in the analysis resuts.

4.0.2 Creation of dummy variables

Now that most of the pre processing is over the last thing left to do is to convert the categorical variables into their respective dummy variables but creating a feature for each category group in the categorical feature. As our categorical variables are in the form of numbers and not strings we need not convert them to numbers, but it is required that we create dummy variables for each categorical group. As the categorical features in our data set are nominal categorical data and not ordinal categories the model shouldn't give higher precedence to a higher number. For example, the categories present in the feature seasons is 1, 2, 3 and 4 which corresponds to spring, summer, fall and winter respectively. As the categories are numeric, the model would give a higher precedence to 4 as it is numerically greater than the rest. Which is not so as they are not ordinal categories and each number refers to a specific season. Hence each group is converted to a binary category. Where 1 imply the presence of that group and 0 represents absence. To make it less complex the first dummy variable is dropped, as when all the other groups are absent it imply that the dropped group is present. This is achieved by running the command in listing 4.1.

```
*******************************************************************
statistical properities of Transportation_expense:
DescribeResult(nobs=740, minmax=(118.0, 378.0), mean=220.62567567567567, variance=4373.354955564495, skewnes
s=0.3756674357536574, kurtosis=-0.33770597431634375)
*******************************************************************
statistical properities of Distance_from_Residence_to_Work:
DescribeResult(nobs=740, minmax=(5.0, 52.0), mean=29.63108108108108, variance=220.1302911655638, skewness=0
.3114498278842765, kurtosis=-1.261270969913851)
*******************************************************************
statistical properities of Service_time:
DescribeResult(nobs=740, minmax=(1.0, 24.0), mean=12.445945945945946, variance=17.386790037669606, skewness=
-0.34301872345212286, kurtosis=-0.19060374758858512)
*******************************************************************
statistical properities of Age  :
DescribeResult(nobs=740, minmax=(27.0, 53.0), mean=36.14054054054054, variance=37.3712906411472, skewness=0
.4894588639031262, kurtosis=-0.25563936337912363)
*******************************************************************
statistical properities of Work_load_Average/day_:
DescribeResult(nobs=740, minmax=(206.0, 343.0), mean=266.94594594594594, variance=1026.0106060051933, skewne
ss=0.5610733905805646, kurtosis=-0.18089326615469226)
*******************************************************************
statistical properities of Hit_target:
DescribeResult(nobs=740, minmax=(87.0, 100.0), mean=94.92297297297297, variance=9.47849723878141, skewness=-
0.44564951370083833, kurtosis=-0.3920431683313459)
*******************************************************************
statistical properities of Weight:
DescribeResult(nobs=740, minmax=(56.0, 108.0), mean=79.03513513513514, variance=165.9771129722415, skewness=
0.016966890485138695, kurtosis=-0.9158612723351429)
*******************************************************************
statistical properities of Height:
DescribeResult(nobs=740, minmax=(165.0, 175.0), mean=170.20675675675676, variance=3.7014427824306027, skewne
ss=-0.44993040994803707, kurtosis=0.7658511236420802)
*******************************************************************
statistical properities of Body_mass_index:
DescribeResult(nobs=740, minmax=(19.0, 38.0), mean=26.681081081081082, variance=18.374472442672715, skewness
=0.3001649161585004, kurtosis=-0.3226064788195093)
*******************************************************************
statistical properities of Absenteeism_time_in_hours:
DescribeResult(nobs=740, minmax=(0.0, 16.0), mean=4.425675675675675, variance=11.790138243791832, skewness=1
.103093333423475, kurtosis=1.2887445861552127)
*******************************************************************
```

Figure 4.1: Skewtest after outlier analysis shows us that the target Absenteeism_time_in_hours still has a large skew of 1.1 when compared to the other features.

Listing 4.1: Creation of dummy variables.

```

## Creating dummy variables
# Where x is all the features in our analysis.

X_lr = pd.get_dummies(X, columns = obj_dtype, drop_first = True)
```

```
*******************************************************************
statistical properities of Transportation_expense:
DescribeResult(nobs=740, minmax=(118.0, 378.0), mean=220.62567567567567, variance=4373.354955564495, skewnes
s=0.3756674357536574, kurtosis=-0.33770597431634375)
*******************************************************************
statistical properities of Distance_from_Residence_to_Work:
DescribeResult(nobs=740, minmax=(5.0, 52.0), mean=29.63108108108108, variance=220.13029111655638, skewness=0
.3114498278842765, kurtosis=-1.261270969913851)
*******************************************************************
statistical properities of Service_time:
DescribeResult(nobs=740, minmax=(1.0, 24.0), mean=12.445945945945946, variance=17.386790037669606, skewness=
-0.34301872345212286, kurtosis=-0.19060374758858512)
*******************************************************************
statistical properities of Age  :
DescribeResult(nobs=740, minmax=(27.0, 53.0), mean=36.14054054054054, variance=37.37129064111472, skewness=0
.4894588639031262, kurtosis=-0.25563936337912363)
*******************************************************************
statistical properities of Work_load_Average/day_:
DescribeResult(nobs=740, minmax=(206.0, 343.0), mean=266.94594594594594, variance=1026.0106060051933, skewne
ss=0.5610733905805646, kurtosis=-0.18089326615469226)
*******************************************************************
statistical properities of Hit_target:
DescribeResult(nobs=740, minmax=(87.0, 100.0), mean=94.92297297297297, variance=9.47849723878141, skewness=-
0.44564951370083833, kurtosis=-0.3920431683313459)
*******************************************************************
statistical properities of Weight:
DescribeResult(nobs=740, minmax=(56.0, 108.0), mean=79.03513513513514, variance=165.9771129722415, skewness=
0.016966890485138695, kurtosis=-0.9158612723351429)
*******************************************************************
statistical properities of Height:
DescribeResult(nobs=740, minmax=(165.0, 175.0), mean=170.20675675675676, variance=3.7014427824306027, skewne
ss=-0.44993040994803707, kurtosis=0.7658511236420802)
*******************************************************************
statistical properities of Body_mass_index:
DescribeResult(nobs=740, minmax=(19.0, 38.0), mean=26.681081081081082, variance=18.374472442672715, skewness
=0.3001649161585004, kurtosis=-0.3226064788195093)
*******************************************************************
statistical properities of Absenteeism_time_in_hours:
DescribeResult(nobs=740, minmax=(0.0, 2.833213344056216), mean=1.4873456789490085, variance=0.43677027019868
41, skewness=-0.23144444623756816, kurtosis=-0.4684714403096022)
*******************************************************************
```

Figure 4.2: Skew test after performing log transform on Absenteeism_time_in_hours that the skewness dropped to -0.23 for Absenteeism_time_in_hours.

Chapter 5

Modeling

5.1 Model Selection

It has be noted in previous stages of our analysis that for different combinations of the independent variables, the count is different. It can be seen that the dependent variable is a continuous variable and hence the type model of model that would be developed for this problem is a regression model. It can also bee seen that there are many features which may contribute to our regression model. That would make this a multivariant regression problem. However the data is not nominally distributed, hence if we are not able to generate a model with good performance we may bin the target veriable and perform multi class classification on it.

5.2 Methodology:

Model Evaluation is an integral part of the model development process as it helps us find the best model for representing our data. It also helps to evaluate as to how it would perform on new data. In order to develop an efficient and accurate model to predict our target variable we shall use a combination of three different methods The three different methods used in our analysis is given below.

- Hold-Out Method
- Hyper parameter Tunning
- Cross-Validation Technique

5.2.1 Hold-Out Method

As evaluating model performance on training data set may lead to develop an over fitted model. Due to this is is required to test the model on a separate data set. Hence the original data set is split into training and testing data. The training data set is used to build a predictive model and the testing data is used to evaluate the model performance.

5.2.2 Hyper parameter Tunning

Hyper parameter Tuning is a method in which the parameters of the model is to be set before training. Scikit-Learn implements a set of sensible default hyper parameters but it may not be optimal. In our analysis we shall use two types of hyper parameter tuning methodology which are *Random Search CV* and *Grid Search CV*. The major difference between them is the run time. As randomized search is drastically lower than grid search. Random search is faster than grid search as we do not provide a set of discrete values to be searched. Rather we provide a statistical distribution for each parameter from which values may be randomly sampled. Where as in grid search a discrete set of values are provided for each parameter and several models are built on each of its combinations which makes it very laboursome.

5.2.3 K-Fold Cross Validation Technique

In K fold cross validation technique the data is divided into k subsets of equal size. We build the model K times, each time leaving out a subset from training. This sub set which was left out will be used for testing. This method helps us develop an unbiased estimate of the model performance.

5.3 Regression Model Building

An ordinary least square linear regression model is first built on the entire data, in order to check for multicollinearity. Before the model is built in multivariate linear regression it is required to do some processing on the data before hand. The equation around which the multivariate regression in statsmodels works is shown below.

$$y = \beta_0 + \beta_1 * X_1 + \beta_2 * X_2 + ... + \beta_n * X_n \tag{5.1}$$

Where β_1, β_2 till β_n are the coefficients for columns X_1, X_2 till X_n respectively and β_0 represents the intercept. Unfortunately in statsmodel the β_0 column is not added by default to out model and hence we are required to add a column with a constant value of one to the data set. Now that this column is added and the dummy variables are created we can go ahead with building the model. It can be noted from the summary of the model that almost non of the features are significnt. This could be due to multicollinearity present in the model. Hence backward elemination is performed in order to remove the multicollinearity present in the model.

5.3.1 Backward Elemination:

In backward elimination, we start with all the features and removes the least significant feature at each iteration which improves the performance of the model. In this procedure we keep an eye on the adjusted r squared value. As we remove insignificant features the model performance increases and the adjusted r square also increases. The iteration is stopped when the value of adjusted r squared drops when compared to the previous iteration. When the adjusted r squared dropped it means that all the features are statistically significant for our model and removal of a single feature beyond this point would reduce the performance of our model. We start the backward elimination process with a total of 69 features with a r square of 0.52 and adjusted are squared value of 0.488.

After 31 iterations it can be noted that the adj r squared has risen to 0.99 and on the 32nd the adjusted R squared value drops to 0.498. Therefor we stop the backward elimination at the 31st iteration. After conducting backward elimination it can be see that the number of features have dropped from 69 to 37.

Multivariate Linear Regression Model after backward elimination

We Start building our model by using the simplest model to the most complex model. Therefor we start our model building using the multiple linear regression model. It can be seen from figure 5.1 that the **Adjusted R-squared is 0.41** which is not impressive, as **it means 41% of the variance of the data is explained by the model.** The code to develop the model and its output can be seen in figure 5.1.

Multivariant linear regression

```
In [1351]: from sklearn.cross_validation import train_test_split
           #Divide data into train and test
           train, test = train_test_split(df, test_size=0.2)

In [1352]: # Train the model using the training sets
           model = sm.OLS(train.iloc[:,36], train.iloc[:,0:36]).fit()

           # make the predictions by the model
           predictions_LR = model.predict(test.iloc[:,0:36])

           r2 = r2_score(test.iloc[:,36], predictions_LR)
           mse = mean_squared_error(test.iloc[:,36], predictions_LR)

           print('Linear Regression Model Performance:')
           print('R-squared = {:0.2}.'.format(r2))
           print('MSE = ',mse)

           Linear Regression Model Performance:
           R-squared = 0.41.
           MSE =  0.24742723773384243
```

Figure 5.1: Multivariant linear regression model after backward elimination with R^2 value 0.41 and MSE 0.24.

Decision Tree Regressor after backward elimination

Now a decision tree regressor model is built on to our data set. At first a model is built on the default parameters of the Decision tree regressor. The default values for the parameters controlling the size of the trees such as max_depth, min_samples_leaf, etc lead to fully grown and unpruned trees which could require large memory consumption and increase the model complexity. It can also cause overfitting of the model. To reduce the complexity and memory consumption the size of the trees should be controlled by setting those parameter values. In our analysis we are going to tune the max_depth parameter which controls the maximum depth of the tree. The default model without any tuning is taken as the base line model.

After this a grid of intervals of 5 is taken from 5 - 50 for max_depth are given to random search cv. The k fold cross validation parameter given is K is 5 and n_iter is 5, which the cross validation is to be repeated 5 times. The optimal depth from random search cv is 10. Now that we narrowed down our search criteria, grid search cv can be conducted on range 5 - 20 max_depth with an interval of 2.

Here it was found that the best parameters for max_depth is 7. The code to develop the models and their output can be seen in figure 5.2, 5.3 & 5.4.

Decision tree

In [464]:
```
#dropping column b0
train = train.drop(["b0"],axis = 1)
test = test.drop(["b0"],axis = 1)
```

In [465]:
```
#Decision Regressor

from sklearn.tree import DecisionTreeRegressor

X_train = train.iloc[:,0:35]
y_train = train.iloc[:,35]

X_test = test.iloc[:,0:35]
y_test = test.iloc[:,35]

DT_reg = DecisionTreeRegressor(random_state=0).fit(X_train,y_train)

DT_reg.get_params()

#Apply model on test data
predictions_DT = DT_reg.predict(X_test)

#R^2
DT_r2 = r2_score(y_test, predictions_DT)
#Calculating MSE
DT_mse = mean_squared_error(y_test, predictions_DT)

print('Decision Regressor Model Performance:')
print('Default Parameters = ',DT_reg.get_params())
print('R-squared = {:0.2}.'.format(DT_r2))
print('MSE = ',DT_mse)
```

```
Decision Regressor Model Performance:
Default Parameters =  {'criterion': 'mse', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': Non
'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
n_weight_fraction_leaf': 0.0, 'presort': False, 'random_state': 0, 'splitter': 'best'}
R-squared = 0.038.
MSE =  0.4015202539881518
```

Figure 5.2: Default Decission tree model with R^2 value 0.038 and MSE 0.4

Random Forest Regressor after backward elemination

Random forest is an ensemble that consists of many decision trees At first a model is built on the default parameters of the Random Forest regressor. To reduce the complexity and memory consumption the max_depth and n_estimators parameters is tuned n_estimators tell the amount of trees to be used in the model. The default model without any tuning is taken as the base line model.

After this a grid of intervals of 2 is taken from 1 - 20 max_depth and a grid of intervals of 2 is taken from 1 - 100 for n_estimators given to random search cv. The k fold cross validation parameter given is K is 5 and n_iter is 5, which the cross validation is to be repeated 5 times. The optimal depth from random search cv is 9 and n_estimators is 15. Now that we narrowed down our search criteria, grid search cv can be conducted on 5 - 15 for max_depth and intervals of 1 is taken from 11 - 20 for n_estimators. Here it was found that the best parameters for max_depth is 12 and n_estimators is 16. The code to develop the models and their output can be seen in figure 5.5, 5.6 & 5.7.

```
In [466]: ##Random Search CV
          from sklearn.model_selection import RandomizedSearchCV
          np.random.seed(0)
          RDT = DecisionTreeRegressor(random_state = 0)
          depth = list(range(5,50,5))
          # Create the random grid
          randDT_grid = {'max_depth': depth}

          randomcv_DT = RandomizedSearchCV(RDT, param_distributions = randDT_grid, n_iter = 5, cv = 5, rando
          randomcv_DT = randomcv_DT.fit(X_train,y_train)

          predictions_RDT = randomcv_DT.predict(X_test)

          view_best_params_RDT = randomcv_DT.best_params_

          best_model = randomcv_DT.best_estimator_

          predictions_RDT = best_model.predict(X_test)

          #R^2
          RDT_r2 = r2_score(y_test, predictions_RDT)
          #Calculate MSE
          RDT_mse = mean_squared_error(y_test, predictions_RDT)

          print('Random Search CV Decision Regressor Model Performance:')
          print('Best Parameters = ',view_best_params_RDT)
          print('R-squared = {:0.2}.'.format(RDT_r2))
          print('MSE = ',RDT_mse)
          print('*******************************************')

          Random Search CV Decision Regressor Model Performance:
          Best Parameters =  {'max_depth': 10}
          R-squared = 0.21.
          MSE =  0.32953047191433227
          *******************************************
```

Figure 5.3: Random search CV Decission tree model with a depth of 10, having R^2 value 0.21 and MSE 0.32.

Gradient Boosting after backward elemination

Gradient boosting is an ensamble boosting method in which a collection of regressors are built in series.It is a method of converting a sequence of weak learners to a complex model. Each regressors prediction is based on the previous regressors by adding weights accordingly. We shall now implement the Gradint boosting model on our data. At first a model is built on the default parameters of the Random Forest regressor. To reduce the complexity and memory consumption the max_depth and n_estimators parameters is tuned n_estimators tell the amount of trees to be used in the model. The default model without any tuning is taken as the base line model.

After this a grid of intervals of 2 is taken from 1 - 10 max_depth and a grid of intervals of 10 is taken from 50 - 150 for n_estimators given to random search cv. The k fold cross validation parameter given is K is 5 and n_iter is 5, which the cross validation is to be repeated 5 times. The optimal depth from random search cv is 3 and n_estimators is 50. Now that we narrowed down our search criteria, grid search cv can be conducted on range 1 - 5 for max_depth and intervals of 5 is taken from 40 - 80 for n_estimators. Here it was found that the best parameters for max_depth is 4 and n_estimators is 45. The code to develop the models and their output can be seen in figure 5.8, 5.9 & 5.10.

```
In [470]: ##Grid Search CV

          from sklearn.model_selection import GridSearchCV


          Gridregr = DecisionTreeRegressor(random_state = 0)
          depth = list(range(1,10,1))

          # Create the grid
          grid_search = {'max_depth': depth}

          ## Grid Search Cross-Validation with 5 fold CV
          gridcv_GDT = GridSearchCV(Gridregr, param_grid = grid_search, cv = 5)
          gridcv_GDT = gridcv_GDT.fit(X_train,y_train)
          view_best_params_GDT = gridcv_GDT.best_params_

          #Apply model on test data
          predictions_GDT = gridcv_GDT.predict(X_test)

          #R^2
          GDT_r2 = r2_score(y_test, predictions_GDT)
          #Calculate MSE
          GDT_mse = mean_squared_error(y_test, predictions_GDT)
          #Calculate MAPE
          GDT_mape = MAPE(y_test, predictions_GDT)

          print('Grid Search CV Decision Regressor Model Performance:')
          print('Best Parameters = ',view_best_params_GDT)
          print('R-squared = {:0.2}.'.format(GDT_r2))
          print('MSE = ',(GDT_mse))
          print('******************************************')

          Grid Search CV Decision Regressor Model Performance:
          Best Parameters =  {'max_depth': 7}
          R-squared = 0.18.
          MSE =  0.34127194268942496
          ******************************************
```

Figure 5.4: Grid search CV Decission tree model with a depth of 7, having R^2 value 0.18 and MSE 0.34.

```
In [477]: from sklearn.ensemble import RandomForestRegressor

          RF_reg = RandomForestRegressor(random_state=0).fit(X_train,y_train)
          RF_reg.get_params()

          #Apply model on test data
          predictions_RF = RF_reg.predict(X_test)

          #R^2
          RF_r2 = r2_score(y_test, predictions_RF)
          #Calculating MSE
          RF_mse = np.mean(( y_test - predictions_RF)**2)

          print('Random Forest Regressor Model Performance:')
          print('Default Parameters = ',RF_reg.get_params())
          print('R-squared = {:0.2}.'.format(RF_r2))
          print('MSE = ',RF_mse)
          print('*********************************************')

          Random Forest Regressor Model Performance:
          Default Parameters =  {'bootstrap': True, 'criterion': 'mse', 'max_depth': None, 'max_features': 'auto', 
          x_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'mi
          amples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 10, 'n_jobs': 1, 'oob_score': False, '
          dom_state': 0, 'verbose': 0, 'warm_start': False}
          R-squared = 0.23.
          MSE =  0.3230720690534794
          *********************************************
```

Figure 5.5: Default Random forest model with R^2 value 0.23 and MSE 0.32

```
In [473]: ##Random Search CV
from sklearn.model_selection import RandomizedSearchCV

RRF = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
             'max_depth': depth}

randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_s
randomcv_rf = randomcv_rf.fit(X_train,y_train)
predictions_RRF = randomcv_rf.predict(X_test)

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(X_test)


#R^2
RRF_r2 = r2_score(y_test, predictions_RRF)
#Calculating MSE
RRF_mse = np.mean(( y_test - predictions_RRF)**2)

print('Random Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RRF)
print('R-squared = {:0.2}.'.format(RRF_r2))
print('MSE = ',RRF_mse)

Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.29.
MSE =  0.29844963753444115
```

Figure 5.6: Random search CV Random forest model with a depth of 9 and n estimators 15. Having R^2 value 0.29 and MSE 0.29.

5.3.2 Principal componant analysis:

Since non of the above models give us a suitable rsquared value. We now perform PCA on our data to reduce the complexity of the data and to make sure that all the variables are independent of each other. Principal component analysis is a technique for feature extraction. In which our data is combined in a specific way, then we can drop the "least important" variables while still retaining the most valuable parts of all of the variables. After performing PCA on the data we can see that the First Principal component explains about 35% of the variance and 20 principal components explain 98.01% of the variance. This can be seen in figure 5.11 & 5.12

Multivariate Linear Regression Model after backward elimination and PCA

We Start building our model by using 20 principal components which explains 98% of the variance of the target variable. It can be seen from figure 5.13 that the **Adjusted R-squared is 0.28** which is not impressive, as **it means 28% of the variance of the data is explained by the model.** The code to develop the model and its output can be seen in figure 5.13.

Decision Tree Regressor after backward elemination and PCA

Now a decision tree regressor model is built on to our data set. At first a model is built on the default parameters of the Decision tree regressor. After this a grid of intervals of 5 is taken from 5 - 50 for

```
In [476]: ## Grid Search CV
          from sklearn.model_selection import GridSearchCV

          regr = RandomForestRegressor(random_state = 0)
          n_estimator = list(range(11,20,1))
          depth = list(range(5,15,2))

          # Create the grid
          grid_search = {'n_estimators': n_estimator,
                         'max_depth': depth}

          ## Grid Search Cross-Validation with 5 fold CV
          gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 5)
          gridcv_rf = gridcv_rf.fit(X_train,y_train)
          view_best_params_GRF = gridcv_rf.best_params_

          #Apply model on test data
          predictions_GRF = gridcv_rf.predict(X_test)

          #R^2
          GRF_r2 = r2_score(y_test, predictions_GRF)
          #Calculating MSE
          GRF_mse = np.mean(( y_test - predictions_GRF)**2)

          print('Grid Search CV Random Forest Regressor Model Performance:')
          print('Best Parameters = ',view_best_params_GRF)
          print('R-squared = {:0.2}.'.format(GRF_r2))
          print('MSE = ',(GRF_mse))

          Grid Search CV Random Forest Regressor Model Performance:
          Best Parameters =  {'max_depth': 7, 'n_estimators': 18}
          R-squared = 0.29.
          MSE =  0.2979419859804356
```

Figure 5.7: Grid search CV Random forest model with a depth of 7 n estimators 18. Having R^2 value 0.29 and MSE 0.29.

max_depth are given to random search cv. The k fold cross validation parameter given is K is 5 and n_iter is 5, which the cross validation is to be repeated 5 times. The optimal depth from random search cv is 10. Now that we narrowed down our search criteria, grid search cv can be conducted on range 1 - 15 max_depth with an interval of 1. Here it was found that the best parameters for max_depth is 5. The code to develop the models and their output can be seen in figure 5.14, 5.15 & 5.16.

Random Forest Regressor after backward elemination and PCA

Random forest is an ensemble that consists of many decision trees At first a model is built on the default parameters of the Random Forest regressor. To reduce the complexity and memory consumption the max_depth and n_estimators parameters is tuned n_estimators tell the amount of trees to be used in the model. The default model without any tuning is taken as the base line model.

After this a grid of intervals of 2 is taken from 1 - 20 max_depth and a grid of intervals of 2 is taken from 1 - 100 for n_estimators given to random search cv. The k fold cross validation parameter given is K is 5 and n_iter is 5, which the cross validation is to be repeated 5 times. The optimal depth from random search cv is 9 and n_estimators is 15. Now that we narrowed down our search criteria, grid search cv can be conducted on 5 - 15 for max_depth and intervals of 1 is taken from 11 - 20 for n_estimators. Here it was found that the best parameters for max_depth is 12 and n_estimators is 16. The code to develop the models and their output can be seen in figure 5.17, 5.18

Gradient Boost

```
In [479]: #Gradient Boost
from sklearn.ensemble import GradientBoostingRegressor

gbt = GradientBoostingRegressor(random_state= 0).fit(X_train,y_train)

predictions_gbt = gbt.predict(X_test)

gbt.get_params()

#R^2
GBR_r2 = r2_score(y_test, predictions_gbt)
#Calculate MSE
GBR_mse = mean_squared_error(y_test, predictions_gbt)

print('Gradient Boosting Regressor Model Performance:')
print('Default Parameters = ',gbt.get_params())
print('R-squared = {:0.2}.'.format(GBR_r2))
print('MSE = ',GBR_mse)
print('**************************************************')
```

```
Gradient Boosting Regressor Model Performance:
Default Parameters =  {'alpha': 0.9, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.
: 'ls', 'max_depth': 3, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0,
urity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
ators': 100, 'presort': 'auto', 'random_state': 0, 'subsample': 1.0, 'verbose': 0, 'warm_start': Fa
R-squared = 0.28.
MSE =  0.3004434155173034
**************************************************
```

Figure 5.8: Default Gradient Boosting model with max depth 3 and n estimator 100, has a R^2 value 0.28 and MSE 0.3

& 5.19.

Gradient Boosting after backward eleminationand PCA

Gradient boosting is an ensemble boosting method in which a collection of regressors are built in series.It is a method of converting asequence of weak learners to a complex model. Each regressors prediction is based on th previous regressors by adding weights accordingly. We shall now implement the Gradint boosting model on our data. At first a model is built on the default parameters of the Random Forest regressor. To reduce the complexity and memory consumption the max_depth and n_estimators parameters is tuned n_estimators tell the amount of trees to be used in the model. The default model without any tuning is taken as the base line model.

After this a grid of intervals of 2 is taken from 1 - 10 max_depth and a grid of intervals of 10 is taken from 50 - 150 for n_estimators given to random search cv. The k fold cross validation parameter given is K is 5 and n_iter is 5, which the cross validation is to be repeated 5 times. The optimal depth from random search cv is 3 and n_estimators is 50. Now that we narrowed down our search criteria, grid search cv can be conducted on range 1 - 5 for max_depth and intervals of 5 is taken from 40 - 80 for n_estimators. Here it was found that the best parameters for max_depth is 4 and n_estimators is 45. The code to develop the models and their output can be seen in figure 5.20, 5.21 & 5.22.

Random Search CV Gradient boosting

```
In [483]: ##Random Search CV
rGBR = GradientBoostingRegressor(random_state = 0)
#loss = ['ls','lad','huber','quantile']
n_estimator = list(range(50,150,10))
#max_feat = ['auto','sqrt','log2']
depth = list(range(1,10,2))

# Create the random grid
rand_GBT = {#'loss': loss,
            'n_estimators': n_estimator,
            #'max_features': max_feat,
            'max_depth': depth}

randomcv_gbt = RandomizedSearchCV(rGBR, param_distributions = rand_GBT, n_iter = 5, cv = 5, random_state=(
randomcv_gbt = randomcv_gbt.fit(X_train,y_train)
predictions_GBT = randomcv_gbt.predict(X_test)


view_best_params_GBT = randomcv_gbt.best_params_

#R^2
rGBR_r2 = r2_score(y_test, predictions_GBT)
#Calculate MSE
rGBR_mse = mean_squared_error(y_test, predictions_GBT)

print('Random Search CV Gradient Boosting Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GBT)
print('R-squared = {:0.2}.'.format(rGBR_r2))
print('MSE = ',rGBR_mse)
print('***************************************************')

Random Search CV Gradient Boosting Regressor Model Performance:
Best Parameters =  {'n_estimators': 50, 'max_depth': 3}
R-squared = 0.25.
MSE =  0.31125407863782906
***************************************************
```

Figure 5.9: Random search CV Gradient Boosting model with a depth of 3 and n estimators 50. Having R^2 value 0.25 and MSE 0.31.

5.4 Classification Model Building

The performance parameters of the models built in the previous section was below par. This could be because the variables are not nominally dristributed. Hence the target variable is binned into 4 categories and multi class classification is performed on it. The vin ranges for the classification is such that values absenteeism time in hours of 0 considered as less than one hour as catergory 1, values absenteeism time in hours of 1 considered as one hour as catergory 2, the values absenteeism time in hours of 2-5 considered as half a day of absenteeism as category 3 and the values with absenteeism time in hours of 5-16 is considered as a whole day of absenteeism as category 4, the distribution of the new target variable is shown is figure 5.23

Before building models we conduct chi squared test on the categorical data for feature selection this is shown in figure 5.24. and the normal procedure of scaling is conducted on the data set. According to the chi squared test Day_of_the_week education and pet is dropped along with weight from the correlation analysis.

```
In [484]: ## Gride Search CV

gGBR = GradientBoostingRegressor(random_state=0)
#loss = ['ls','lad','huber','quantile']
n_estimator = list(range(40,80,5))
#max_feat = ['auto','sqrt','log2']
depth = list(range(1,5,1))

# Create the random grid
grid_GBT = {#'loss': loss,
            'n_estimators': n_estimator,
            #'max_features': max_feat,
            'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_GBT = GridSearchCV(gGBR, param_grid = grid_GBT, cv = 5)
gridcv_GBT = gridcv_GBT.fit(X_train,y_train)
view_best_params_gridGRF = gridcv_GBT.best_params_

#Apply model on test data
predictions_gridGBT = gridcv_GBT.predict(X_test)

#R^2
gGBR_r2 = r2_score(y_test, predictions_gridGBT)
#Calculate MSE
gGBR_mse = mean_squared_error(y_test, predictions_gridGBT)

#Calculate MAPE
gGBR_mape = MAPE(y_test, predictions_gridGBT)

print('Grid Search CV Gradient Boosting Regressor Model Performance:')
print('Best Parameters = ',view_best_params_gridGRF)
print('R-squared = {:0.2}.'.format(gGBR_r2))
print('MSE = ', gGBR_mse)
print('**************************************************')

Grid Search CV Gradient Boosting Regressor Model Performance:
Best Parameters =  {'max_depth': 4, 'n_estimators': 45}
R-squared = 0.27.
MSE =  0.3065385845417901
**************************************************
```

Figure 5.10: Grid search CV Gradient Boosting model with a depth of 4 n estimators 45. Having R^2 value 0.27 and MSE 0.3.

Decision tree multiclass classifier

A multiclass decision tree classifier is built on the train data to predict as to predict the class of absenteeism. The code confusion confusion matrix is shown in figure 5.25. The decision tree classifier has an accuracy of 59.7%.

Logistic regression multi class classifier

A Logistic regression multi class classifier model is built on the train data to predict as to predict the class of absenteeism. The code and output is shown in figure 5.26. The Logistic regression classifier has an accuracy of 61.6%.

Random forest multi class classifier

A Random forest multi class classifier model is built on the train data to predict as to predict the class of absenteeism. The code and output is shown in figure 5.27. The classifier has an accuracy of 61.6%.

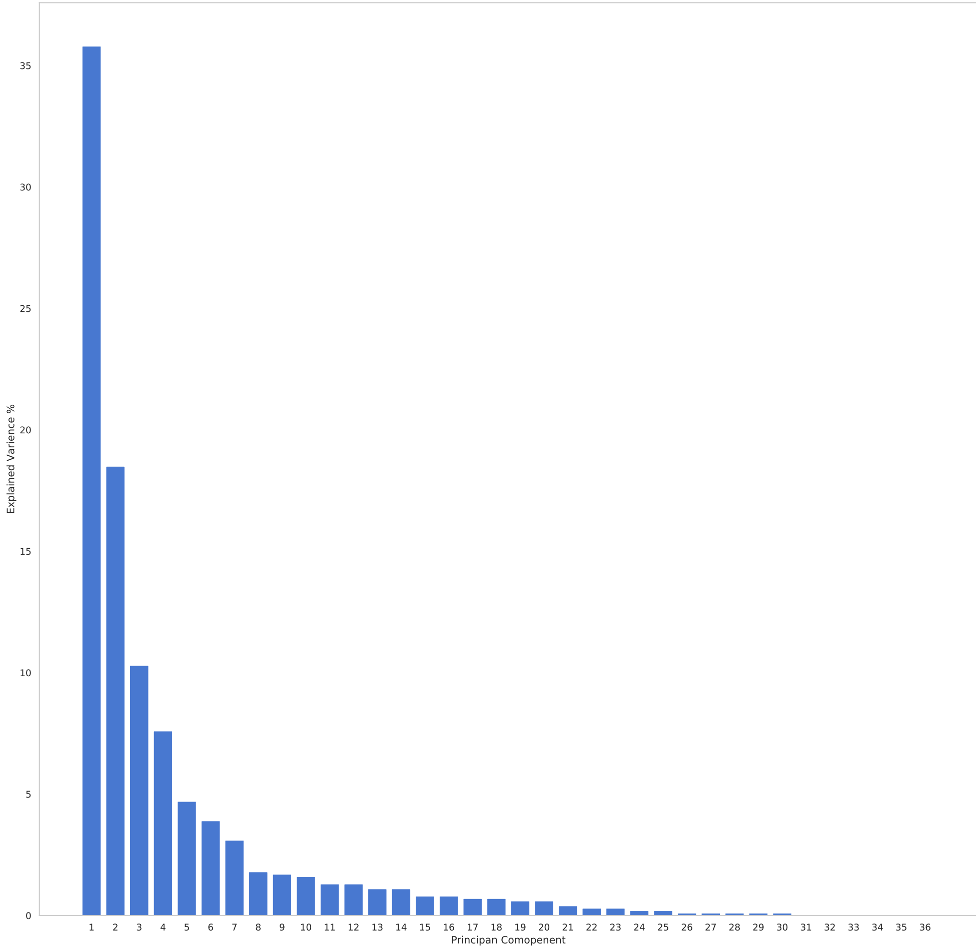


Figure 5.11: Explained variance of each principal component.

KNN classifier

A KNN classifier model is built on the train data to predict as to predict the class of absenteeism. The code and output is shown in figure 5.28. The KNN classifier has an accuracy of 50.6%.

Naive Bayes

Naive Bayes classifier model is built on the train data to predict as to predict the class of absenteeism. The code and output is shown in figure 5.29. The Naive Bayes classifier has an accuracy of 47.3%.s

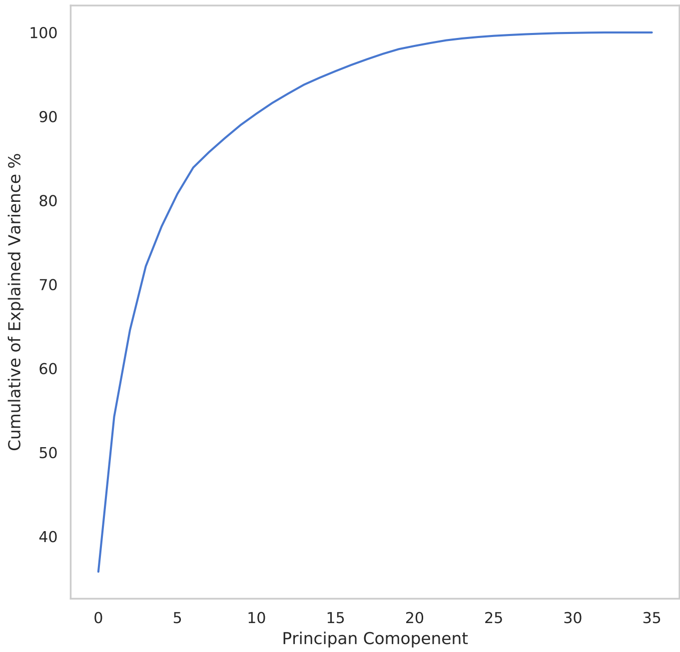


Figure 5.12: Cumulative of explained variance of PCA.

Multivariant linear regression

In [1351]:
```python
from sklearn.cross_validation import train_test_split
#Divide data into train and test
train, test = train_test_split(df, test_size=0.2)
```

In [1352]:
```python
# Train the model using the training sets
model = sm.OLS(train.iloc[:,36], train.iloc[:,0:36]).fit()

# make the predictions by the model
predictions_LR = model.predict(test.iloc[:,0:36])

r2 = r2_score(test.iloc[:,36], predictions_LR)
mse = mean_squared_error(test.iloc[:,36], predictions_LR)

print('Linear Regression Model Performance:')
print('R-squared = {:0.2}.'.format(r2))
print('MSE = ',mse)
```

```
Linear Regression Model Performance:
R-squared = 0.41.
MSE =  0.24742723773384243
```

Figure 5.13: Multivariant linear regression model after backword elimination and PCA with R^2 value 0.28 and MSE 0.29.

```
In [507]: #Decision Regressor

from sklearn.tree import DecisionTreeRegressor

DT_reg = DecisionTreeRegressor(max_depth = 2, random_state=0).fit(X_train,y_train)

DT_reg.get_params()

#Apply model on test data
predictions_DT = DT_reg.predict(X_test)

#R^2
DT_r2 = r2_score(y_test, predictions_DT)
#Calculating MSE
DT_mse = mean_squared_error(y_test, predictions_DT)

print('Decision Regressor Model Performance:')
print('Default Parameters = ',DT_reg.get_params())
print('R-squared = {:0.4}.'.format(DT_r2))
print('MSE = ',DT_mse)
print('*******************************************')
```

```
Decision Regressor Model Performance:
Default Parameters =  {'criterion': 'mse', 'max_depth': 2, 'max_features': None, 'max_leaf_nodes': None,
n_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'm
eight_fraction_leaf': 0.0, 'presort': False, 'random_state': 0, 'splitter': 'best'}
R-squared = 0.1032.
MSE =  0.3744484881210333
*******************************************
```

Figure 5.14: Default Decision tree model with R^2 value 0.10 and MSE 0.37

Decision tree Random Search CV

```
In [508]: ##Random Search CV
from sklearn.model_selection import RandomizedSearchCV
np.random.seed(0)
RDT = DecisionTreeRegressor(random_state = 0)
depth = list(range(5,50,5))
# Create the random grid
randDT_grid = {'max_depth': depth}

randomcv_DT = RandomizedSearchCV(RDT, param_distributions = randDT_grid, n_iter = 5, cv = 5, rando
randomcv_DT = randomcv_DT.fit(X_train,y_train)

predictions_RDT = randomcv_DT.predict(X_test)

view_best_params_RDT = randomcv_DT.best_params_

best_model = randomcv_DT.best_estimator_

predictions_RDT = best_model.predict(X_test)

#R^2
RDT_r2 = r2_score(y_test, predictions_RDT)
#Calculate MSE
RDT_mse = mean_squared_error(y_test, predictions_RDT)

print('Random Search CV Decision Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RDT)
print('R-squared = {:0.2}.'.format(RDT_r2))
print('MSE = ',RDT_mse)
print('*******************************************')
```

```
Random Search CV Decision Regressor Model Performance:
Best Parameters =  {'max_depth': 10}
R-squared = 0.015.
MSE =  0.41137102943099124
*******************************************
```

Figure 5.15: Random search CV Decission tree model with a depth of 10, having R^2 value 0.015 and MSE 0.41.

Decision tree Grid Search CV

```
In [513]: ##Grid Search CV

          from sklearn.model_selection import GridSearchCV


          Gridregr = DecisionTreeRegressor(random_state = 0)
          depth = list(range(1,15,1))

          # Create the grid
          grid_search = {'max_depth': depth}

          ## Grid Search Cross-Validation with 5 fold CV
          gridcv_GDT = GridSearchCV(Gridregr, param_grid = grid_search, cv = 5)
          gridcv_GDT = gridcv_GDT.fit(X_train,y_train)
          view_best_params_GDT = gridcv_GDT.best_params_

          #Apply model on test data
          predictions_GDT = gridcv_GDT.predict(X_test)

          #R^2
          GDT_r2 = r2_score(y_test, predictions_GDT)
          #Calculate MSE
          GDT_mse = mean_squared_error(y_test, predictions_GDT)
          #Calculate MAPE
          GDT_mape = MAPE(y_test, predictions_GDT)

          print('Grid Search CV Decision Regressor Model Performance:')
          print('Best Parameters = ',view_best_params_GDT)
          print('R-squared = {:0.2}.'.format(GDT_r2))
          print('MSE = ',(GDT_mse))
          print('******************************************')

          Grid Search CV Decision Regressor Model Performance:
          Best Parameters =  {'max_depth': 5}
          R-squared = 0.21.
          MSE =  0.3294611813627842
          ******************************************
```

Figure 5.16: Grid search CV Decision tree model with a depth of 5, having R^2 value 0.21 and MSE 0.32.

Random forest

```
In [514]: from sklearn.ensemble import RandomForestRegressor

          RF_reg = RandomForestRegressor(n_estimators = 1000, random_state=0).fit(X_train,y_train)
          RF_reg.get_params()

          #Apply model on test data
          predictions_RF = RF_reg.predict(X_test)

          #R^2
          RF_r2 = r2_score(y_test, predictions_RF)
          #Calculating MSE
          RF_mse = np.mean(( y_test - predictions_RF)**2)

          print('Random Forest Regressor Model Performance:')
          print('Default Parameters = ',RF_reg.get_params())
          print('R-squared = {:0.4}.'.format(RF_r2))
          print('MSE = ',RF_mse)
          print('**********************************************')

          Random Forest Regressor Model Performance:
          Default Parameters =  {'bootstrap': True, 'criterion': 'mse', 'max_depth': None, 'max_features': 'auto'
          x_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, '
          amples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 1000, 'n_jobs': 1, 'oob_score': Fals
          andom_state': 0, 'verbose': 0, 'warm_start': False}
          R-squared = 0.2736.
          MSE =  0.30330721444799214
          **********************************************
```

Figure 5.17: Default Random forest model with R^2 value 0.27 and MSE 0.3

Random forest random Search CV

```
In [515]: ##Random Search CV
from sklearn.model_selection import RandomizedSearchCV

RRF = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
             'max_depth': depth}

randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_s
randomcv_rf = randomcv_rf.fit(X_train,y_train)
predictions_RRF = randomcv_rf.predict(X_test)

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(X_test)


#R^2
RRF_r2 = r2_score(y_test, predictions_RRF)
#Calculating MSE
RRF_mse = np.mean(( y_test - predictions_RRF)**2)

print('Random Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RRF)
print('R-squared = {:0.2}.'.format(RRF_r2))
print('MSE = ',RRF_mse)

Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.26.
MSE =  0.30971718768557643
```

Figure 5.18: Random search CV Random forest model with a depth of 9 and n estimators 15. Having R^2 value 0.26 and MSE 0.3.

Random forest Grid Search CV

In [516]:
```
## Grid Search CV
from sklearn.model_selection import GridSearchCV

regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

# Create the grid
grid_search = {'n_estimators': n_estimator,
               'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 5)
gridcv_rf = gridcv_rf.fit(X_train,y_train)
view_best_params_GRF = gridcv_rf.best_params_

#Apply model on test data
predictions_GRF = gridcv_rf.predict(X_test)

#R^2
GRF_r2 = r2_score(y_test, predictions_GRF)
#Calculating MSE
GRF_mse = np.mean(( y_test - predictions_GRF)**2)

print('Grid Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GRF)
print('R-squared = {:0.2}.'.format(GRF_r2))
print('MSE = ',(GRF_mse))
```

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 16}
R-squared = 0.27.
MSE =  0.3027266645879698
```

Figure 5.19: Grid search CV Random forest model with a depth of 7 n estimators 16. Having R^2 value 0.27 and MSE 0.3.

Gradient Boost

```
In [517]: #Gradient Boost
from sklearn.ensemble import GradientBoostingRegressor

gbt = GradientBoostingRegressor(random_state= 0).fit(X_train,y_train)

predictions_gbt = gbt.predict(X_test)

gbt.get_params()

#R^2
GBR_r2 = r2_score(y_test, predictions_gbt)
#Calculate MSE
GBR_mse = mean_squared_error(y_test, predictions_gbt)

print('Gradient Boosting Regressor Model Performance:')
print('Default Parameters = ',gbt.get_params())
print('R-squared = {:0.2}.'.format(GBR_r2))
print('MSE = ',GBR_mse)
print('**************************************************')
```

```
Gradient Boosting Regressor Model Performance:
Default Parameters =  {'alpha': 0.9, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0
: 'ls', 'max_depth': 3, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0
urity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0
ators': 100, 'presort': 'auto', 'random_state': 0, 'subsample': 1.0, 'verbose': 0, 'warm_start': F
R-squared = 0.26.
MSE =  0.3106926099088413
**************************************************
```

Figure 5.20: Default Gradient Boosting model with max depth 3 and n estimator 100, has a R^2 value 0.26 and MSE 0.3

Random Search CV Gradient boosting

In [518]:

```
##Random Search CV
rGBR = GradientBoostingRegressor(random_state = 0)
#loss = ['ls','lad','huber','quantile']
n_estimator = list(range(50,150,10))
#max_feat = ['auto','sqrt','log2']
depth = list(range(1,10,2))

# Create the random grid
rand_GBT = {#'loss': loss,
            'n_estimators': n_estimator,
            #'max_features': max_feat,
            'max_depth': depth}

randomcv_gbt = RandomizedSearchCV(rGBR, param_distributions = rand_GBT, n_iter = 5, cv = 5, random_state
randomcv_gbt = randomcv_gbt.fit(X_train,y_train)
predictions_GBT = randomcv_gbt.predict(X_test)


view_best_params_GBT = randomcv_gbt.best_params_

#R^2
rGBR_r2 = r2_score(y_test, predictions_GBT)
#Calculate MSE
rGBR_mse = mean_squared_error(y_test, predictions_GBT)

print('Random Search CV Gradient Boosting Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GBT)
print('R-squared = {:0.2}.'.format(rGBR_r2))
print('MSE = ',rGBR_mse)
print('**************************************************')
```

```
Random Search CV Gradient Boosting Regressor Model Performance:
Best Parameters =  {'n_estimators': 60, 'max_depth': 3}
R-squared = 0.28.
MSE =  0.3021934297233443
**************************************************
```

Figure 5.21: Random search CV Gradient Boosting model with a depth of 3 and n estimators 60. Having R^2 value 0.28 and MSE 0.30.

Gradient Boosting Gride Search CV

```
In [519]: ## Gride Search CV

gGBR = GradientBoostingRegressor(random_state=0)
#loss = ['ls','lad','huber','quantile']
n_estimator = list(range(40,80,5))
#max_feat = ['auto','sqrt','log2']
depth = list(range(1,5,1))

# Create the random grid
grid_GBT = {#'loss': loss,
            'n_estimators': n_estimator,
            #'max_features': max_feat,
            'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_GBT = GridSearchCV(gGBR, param_grid = grid_GBT, cv = 5)
gridcv_GBT = gridcv_GBT.fit(X_train,y_train)
view_best_params_gridGRF = gridcv_GBT.best_params_

#Apply model on test data
predictions_gridGBT = gridcv_GBT.predict(X_test)

#R^2
gGBR_r2 = r2_score(y_test, predictions_gridGBT)
#Calculate MSE
gGBR_mse = mean_squared_error(y_test, predictions_gridGBT)

#Calculate MAPE
gGBR_mape = MAPE(y_test, predictions_gridGBT)

print('Grid Search CV Gradient Boosting Regressor Model Performance:')
print('Best Parameters = ',view_best_params_gridGRF)
print('R-squared = {:0.2}.'.format(gGBR_r2))
print('MSE = ', gGBR_mse)
print('**************************************************')
```

```
Grid Search CV Gradient Boosting Regressor Model Performance:
Best Parameters =  {'max_depth': 2, 'n_estimators': 65}
R-squared = 0.31.
MSE =  0.28671413364552173
**************************************************
```

Figure 5.22: Grid search CV Gradient Boosting model with a depth of 4 n estimators 45. Having R^2 value 0.31 and MSE 0.28.

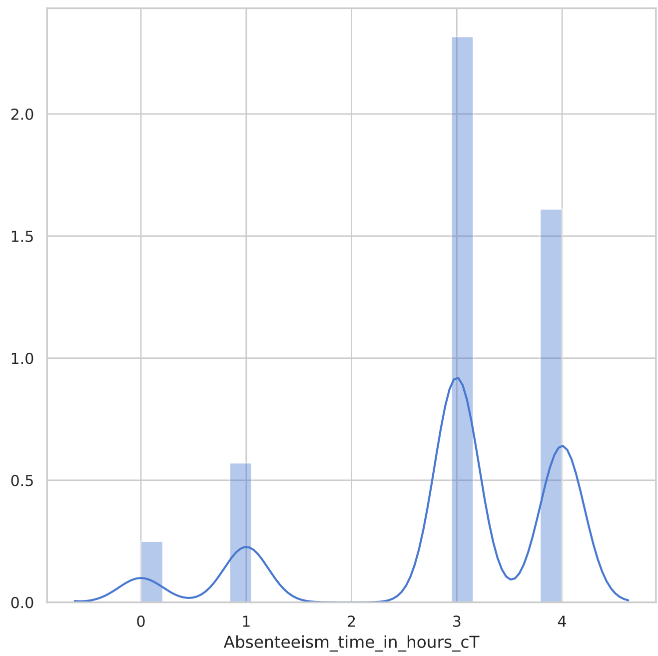


Figure 5.23: The distribution of the new binned target variable.

```
In [525]: from scipy.stats import chi2_contingency
          #loop for chi square values
          for i in obj_dtype:
              print(i)
              chi2, p, dof, ex = chi2_contingency(pd.crosstab(df['Absenteeism_time_in_hours'], df[i]))
              print(p)

ID
7.705346449864873e-10
Reason_for_absence
3.038684341153542e-49
Month_of_absence
5.198801170459447e-09
Disciplinary_failure
1.7506819169177113e-117
Son
2.2268873278599972e-08
Social_drinker
0.011539968647928535
Pet
0.12608552548493424
```

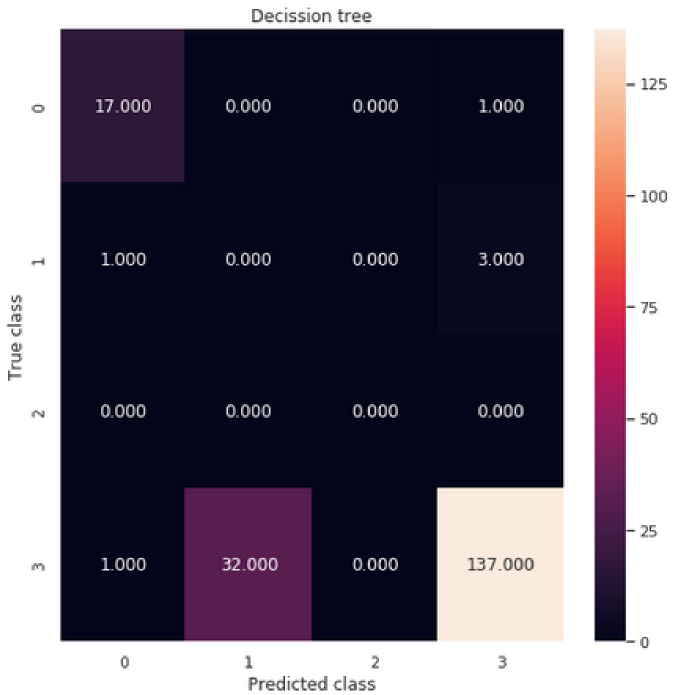
Figure 5.24: Chi squared test.

Decision Tree Classifier

In [542]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0).fit(X_train,y_train)
```

In [543]:
```
y_pred = clf.predict(X_test)
```

In [553]:
```
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import seaborn as sns
forest_cm = metrics.confusion_matrix(y_pred, y_test,[0,1,2,3])
sns.heatmap(forest_cm, annot=True, fmt='.3f' )
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Decission tree')
plt.savefig('dt.pdf')
```



In [545]:
```
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, clf.predict(X_test))))
```

```
Accuracy: 0.597
```

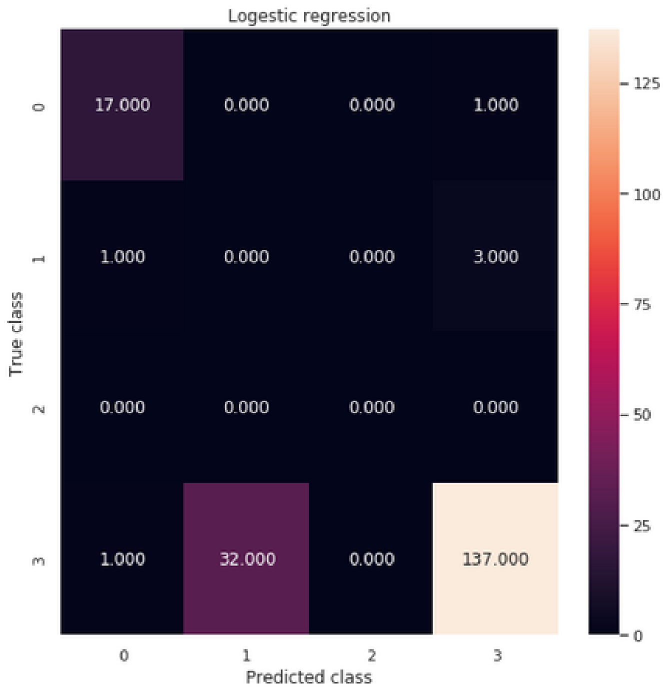
Figure 5.25: Decision tree multiclass classifier with an accuracy of 59.7%.

Logistic Regression Multinomial classifier

In [554]:
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial',max_iter=1000)
clf.fit(X_train, y_train)
y_pred=clf.predict(X_test)

from sklearn.metrics import confusion_matrix
from sklearn import metrics
import seaborn as sns
logr_cm = metrics.confusion_matrix(y_pred, y_test,[0,1,2,3])
sns.heatmap(logr_cm, annot=True, fmt='.3f' )
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Logestic regression')
plt.savefig('Logestic_regression')
```



In [555]:
```
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, clf.predict(X_test))))
```

```
Accuracy: 0.616
```

Figure 5.26: Logistic regression multi class classier with an curacy of 61.5%.

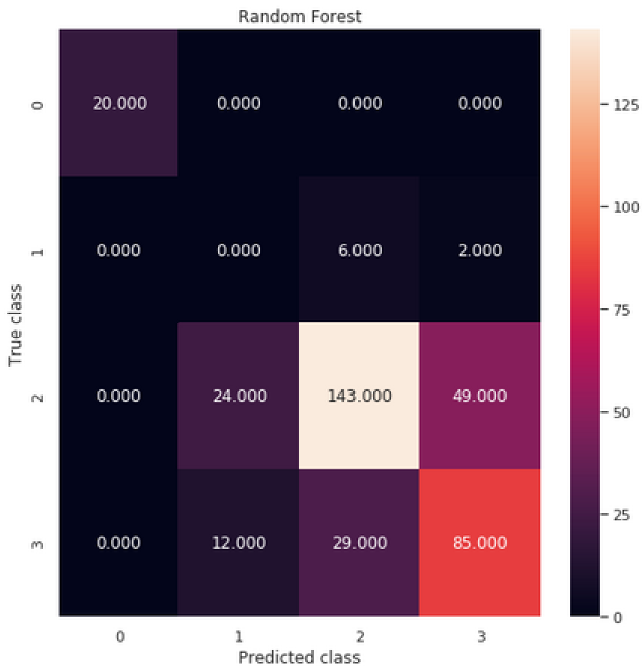
Random Forest Multiclass classifier

In [556]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

RF_model = RandomForestClassifier(n_estimators = 500).fit(X_train, y_train)
```

In [557]:
```
RF_Predictions = RF_model.predict(X_test)
```

In [558]:
```
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import seaborn as sns
rf_cm = metrics.confusion_matrix(y_pred, RF_Predictions)
sns.heatmap(rf_cm, annot=True, fmt='.3f' )
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Random Forest')
plt.savefig('random_forest')
```



In [559]:
```
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, RF_model.predict(X_test))))
```

```
Accuracy: 0.670
```

Figure 5.27: Random forest multiclass classier with an curacy of 67%.

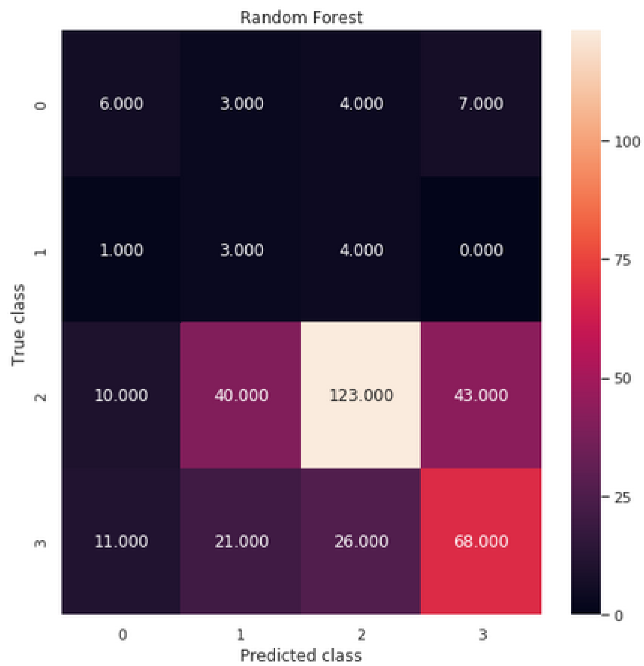
KNN

In [560]:
```
from sklearn.neighbors import KNeighborsClassifier

KNN_model = KNeighborsClassifier(n_neighbors = 3).fit(X_train, y_train)
```

In [561]:
```
#predict test cases
KNN_Predictions = KNN_model.predict(X_test)
```

In [562]:
```
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import seaborn as sns
rf_cm = metrics.confusion_matrix(y_pred, KNN_Predictions)
sns.heatmap(rf_cm, annot=True, fmt='.3f' )
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Random Forest')
plt.savefig('random_forest')
```



In [563]:
```
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, KNN_model.predict(X_test))))
```

```
Accuracy: 0.505
```

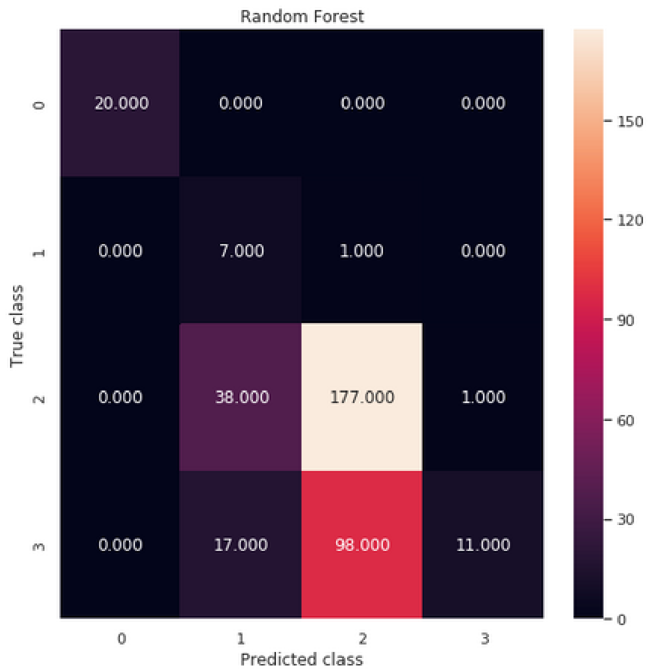
Figure 5.28: KNN classier with an curacy of 50.5%.

Naive Bayes

In [564]:
```
from sklearn.naive_bayes import GaussianNB
#Naive Bayes implementation
NB_model = GaussianNB().fit(X_train, y_train)
```

In [565]:
```
#predict test cases
NB_Predictions = NB_model.predict(X_test)
```

In [566]:
```
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import seaborn as sns
rf_cm = metrics.confusion_matrix(y_pred, NB_Predictions)
sns.heatmap(rf_cm, annot=True, fmt='.3f' )
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Random Forest')
plt.savefig('random_forest')
```



In [567]:
```
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, NB_model.predict(X_test))))
```

```
Accuracy: 0.473
```

Figure 5.29: Naive Bayes classifer with an curacy of 47.3%.

Chapter 6

Conclusion

6.1 Model Evaluation

Now that a few models have been created on our data set, it is required that a suitable evaluation matrix is selected it compare the different model. For the regression model there are a few popular evaluation matrix which are commonly used. These matrix are : MAPE - Mean absolute percentage error, is the measure accuracy as a percentage of error, MSE - Mean squared error, it is the the mean squared errors and RMSE - Root mean squared error, it is the square root of the mean squared errors. For the classification model we would be using accuracy.

MSE is selected as it gives us the goodness of fit of the model for the regression model. The smaller the MSE the better the fit. R squared values is also taken into account for our regression model as it explains as to how much of the variance of the target variable is explained. For the classification model we would be using accuracy.

6.2 Model Selection

Table 6.1: Regression model performance after backward elimination

Model name	MSE	R-squared
Multivariant linear regression	0.247	0.41
Decision tree regressor	0.4	0.038
Random search cv Decision tree	0.329	0.21
Grid search cv Decision tree	0.341	0.18
Random forest	0.32	0.23
Random search cv Random forest	0.29	0.29
Grid search cv Random forest	0.29	0.29
Gradient boosting	0.3	0.28
Random search cv Gradient boosting	0.31	0.25
Grid search cv Gradient boosting	0.30	0.27

From table 6.1, 6.2 & 6.3 we can see that the best model with good performance is the multiclass random forest classifier model. In this model the target variable is binned to form 4 separate categories. Due to this we are able to get an accuracy of 67%. Hence this model is selected.

Table 6.2: Regression model performance after backward elemination and PCA

Model name	MSE	R-squared
Multivariant linear regression	0.29	0.28
Decision tree regressor	0.37	0.10
Random search cv Decision tree	0.41	0.015
Grid search cv Decision tree	0.32	0.21
Random forest	0.3	0.27
Random search cv Random forest	0.3	0.26
Grid search cv Random forest	0.3	0.27
Gradient boosting	0.3	0.26
Random search cv Gradient boosting	0.3	0.28
Grid search cv Gradient boosting	0.28	0.31

Table 6.3: Classification model performance

Model name	Accuracy
Decision tree regressor	59.7%
Logestic regression	61.6%
Random Forest	67%
KNN	50.5%
Naive Nayes	0.47

6.3 What changes company should bring to reduce the number of absenteeism?

From our analysis we were able to see that employees who had only an high school degree tend to be absent more often. It was also noted that people below the age of 30 and having service time less than 8 years tend to be absent often. Also when employees have medium work load i.e. between 250 - 300 average work load per day are often absent and people who have more or lesser work are not absent often. Also people with out children or pets often are absent compared to the rest. Therefor changes to be made is more college graduates to be hired in place of high school degree holders. People above the age of 30 need to be hired and the company need to retain people after they have 16 years of service. While hiring the employer should check to see if the employee or candidate has kids or pets and higher accordingly. Correct amount of work must be given to people. As it is noted people with work load 250-300 are absent often this could imply that the work load is less.

6.4 How much losses every month can we project in 2011 if same trend of absenteeism continues?

As we don't have the data for 2011, we shall use the given data to calculate the loss in 2011 assuming the trend remains same. We shall calculate the loss of work in time (hrs) which would be the total sum of absenteeism time in hours for each month respectively. We shall also calculate the loss in work load. Assuming work load average per day is the target workload for that day we shall calculate its loss due to absenteeism time in hours by the formula given below

$$lossinwork = \frac{work_load/day}{24} * Absenteeisminhours \tag{6.1}$$

Where 24 is the total number of hours in a day. The loss that will be incurred in 2011 is shown in figure 6.1

:

Month_of_absence	Total Absenteeism time in a month (hrs)	Work loss per month
1.0	171.0	2252.833333
2.0	279.0	3164.875000
3.0	452.0	5281.916667
4.0	240.0	2728.875000
5.0	273.0	2794.166667
6.0	249.0	2815.083333
7.0	392.0	4129.750000
8.0	248.0	2438.000000
9.0	205.0	2298.666667
10.0	297.0	3326.708333
11.0	265.0	3136.833333
12.0	204.0	2208.291667

Figure 6.1: The loss that will be incured in 2011

Appendices

Appendix A

R code snapshots:

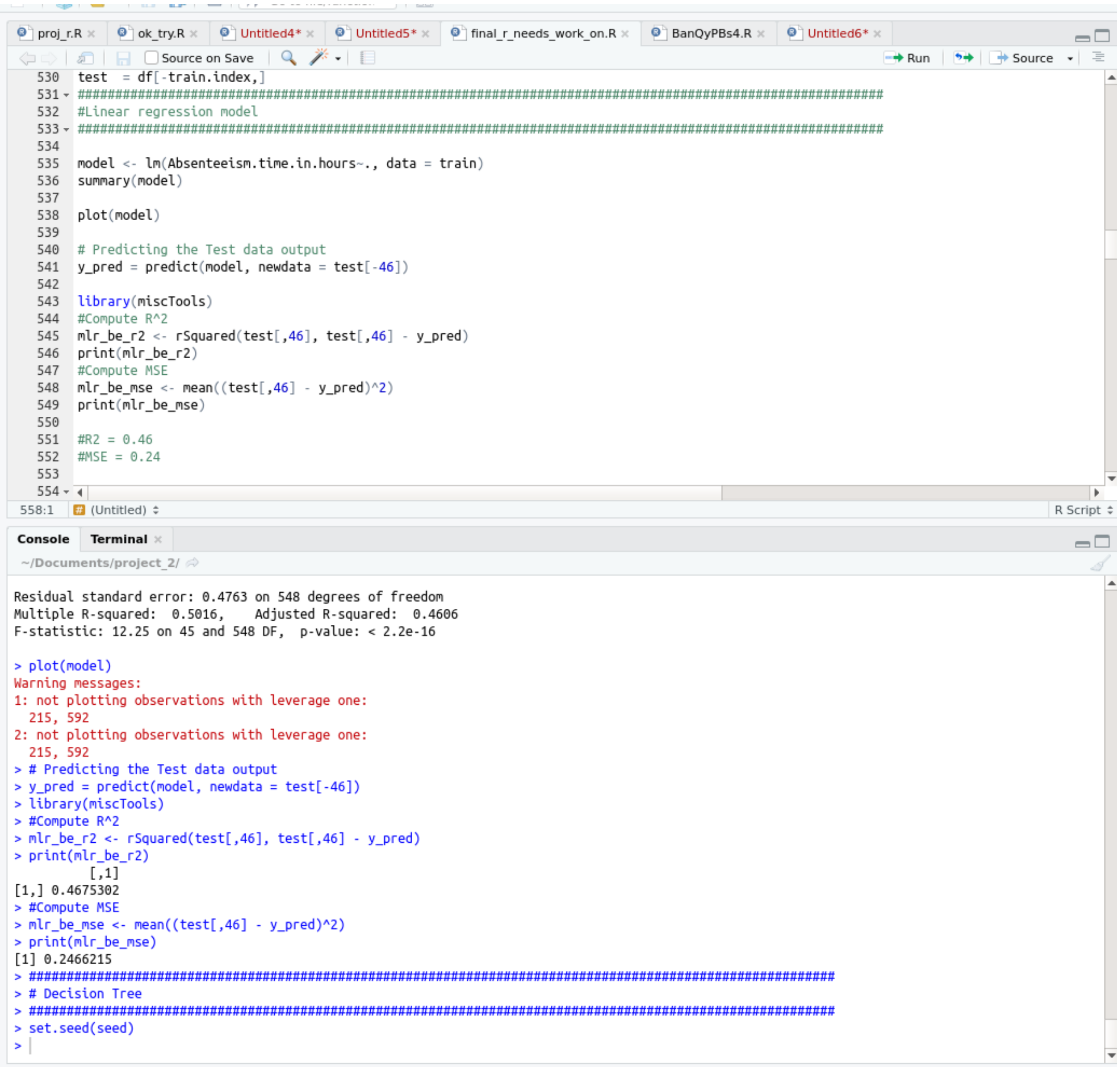


Figure A.1: Linear regression model in r with output.

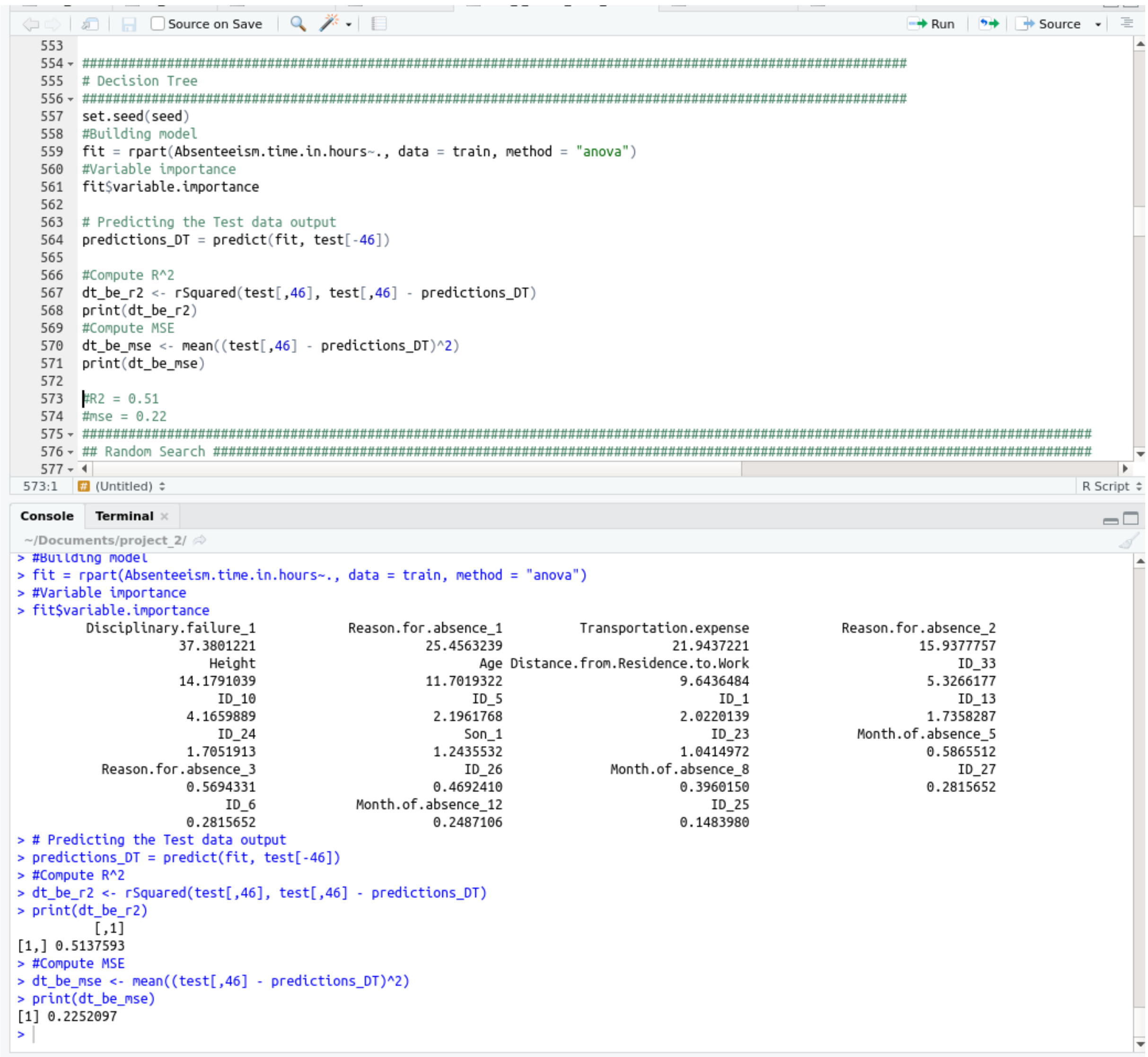


Figure A.2: Decision tree model in r with output

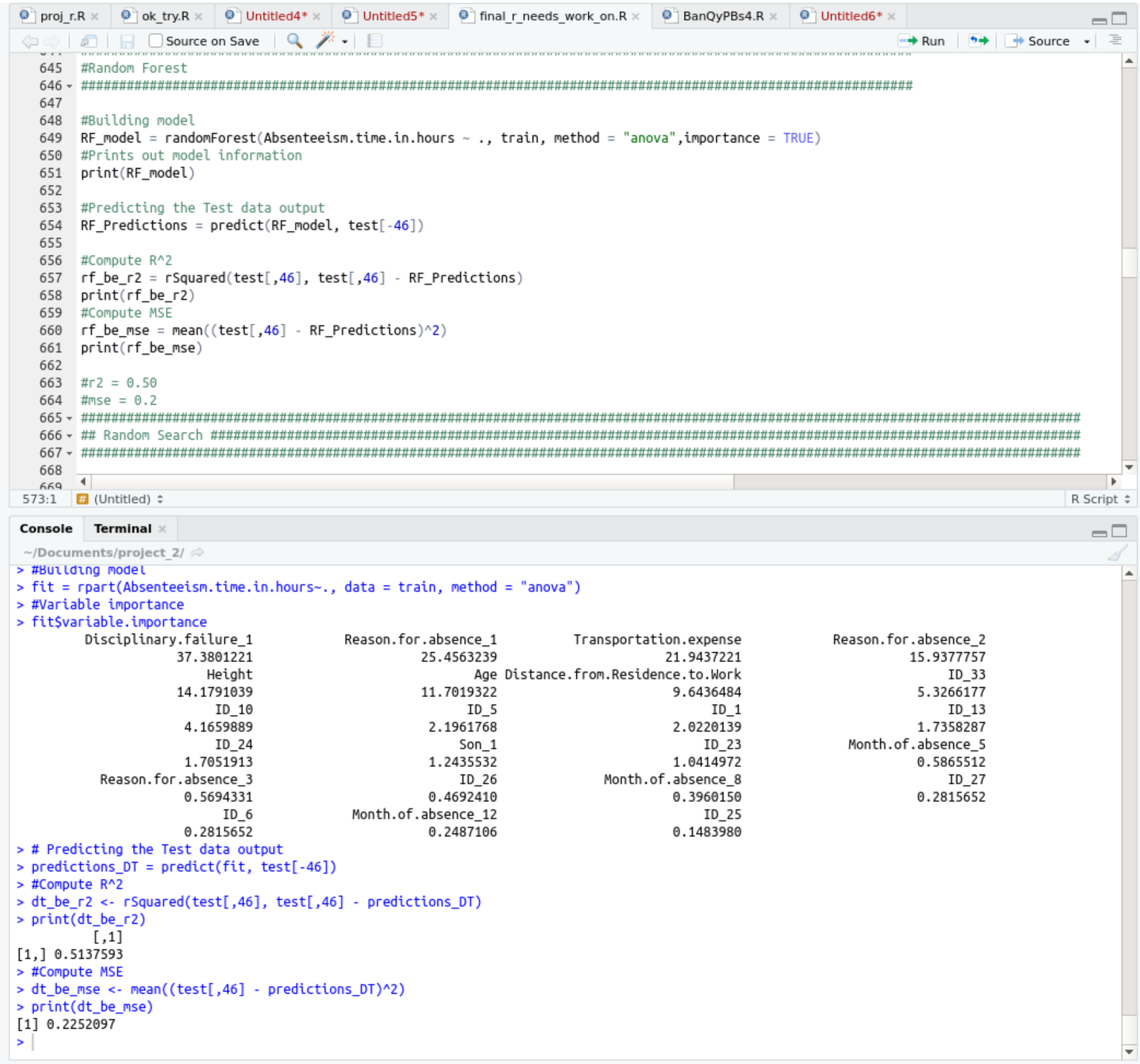


Figure A.3: Randeom forest model in r with output

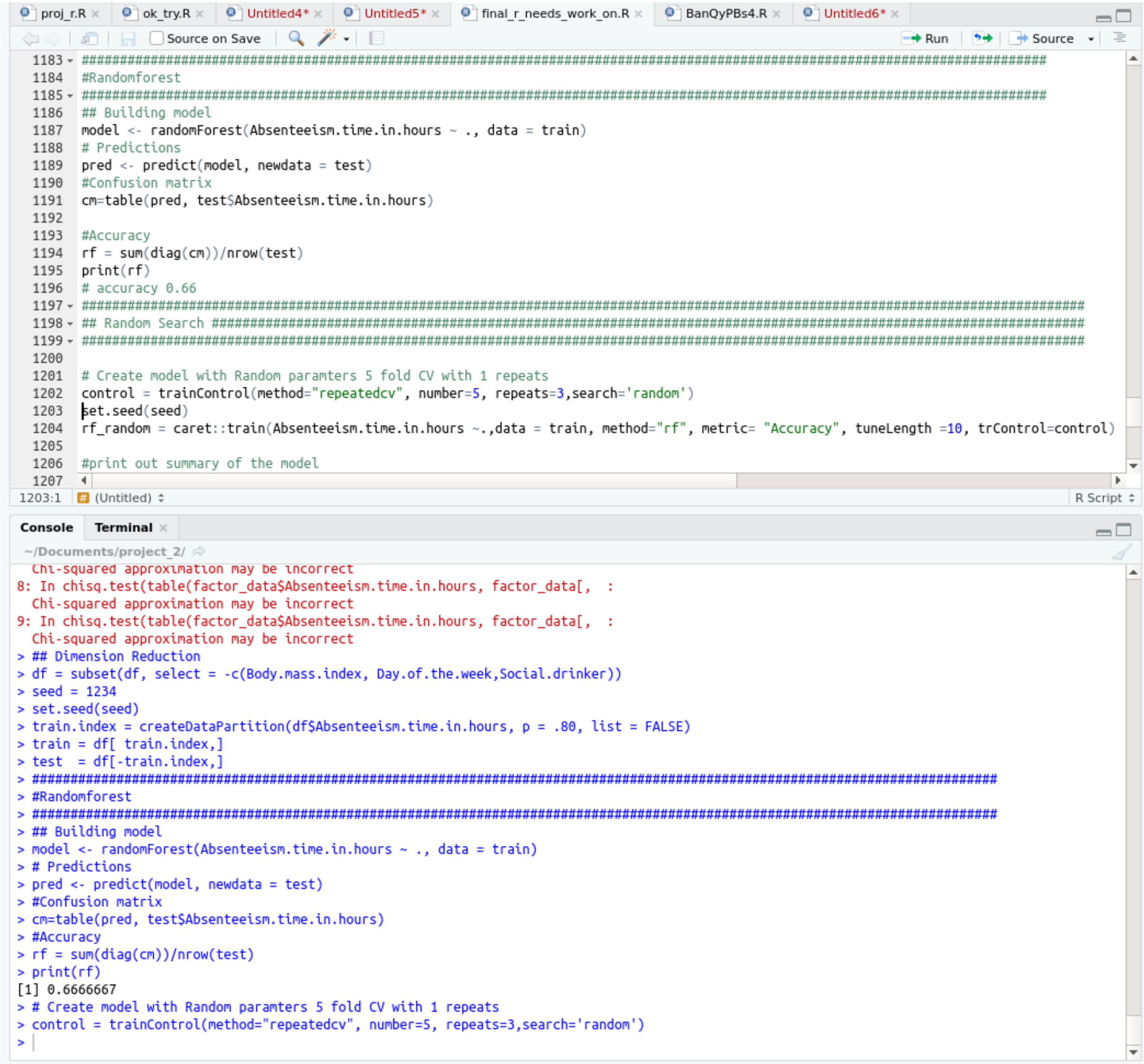


Figure A.4: Randeom forest classifier in r

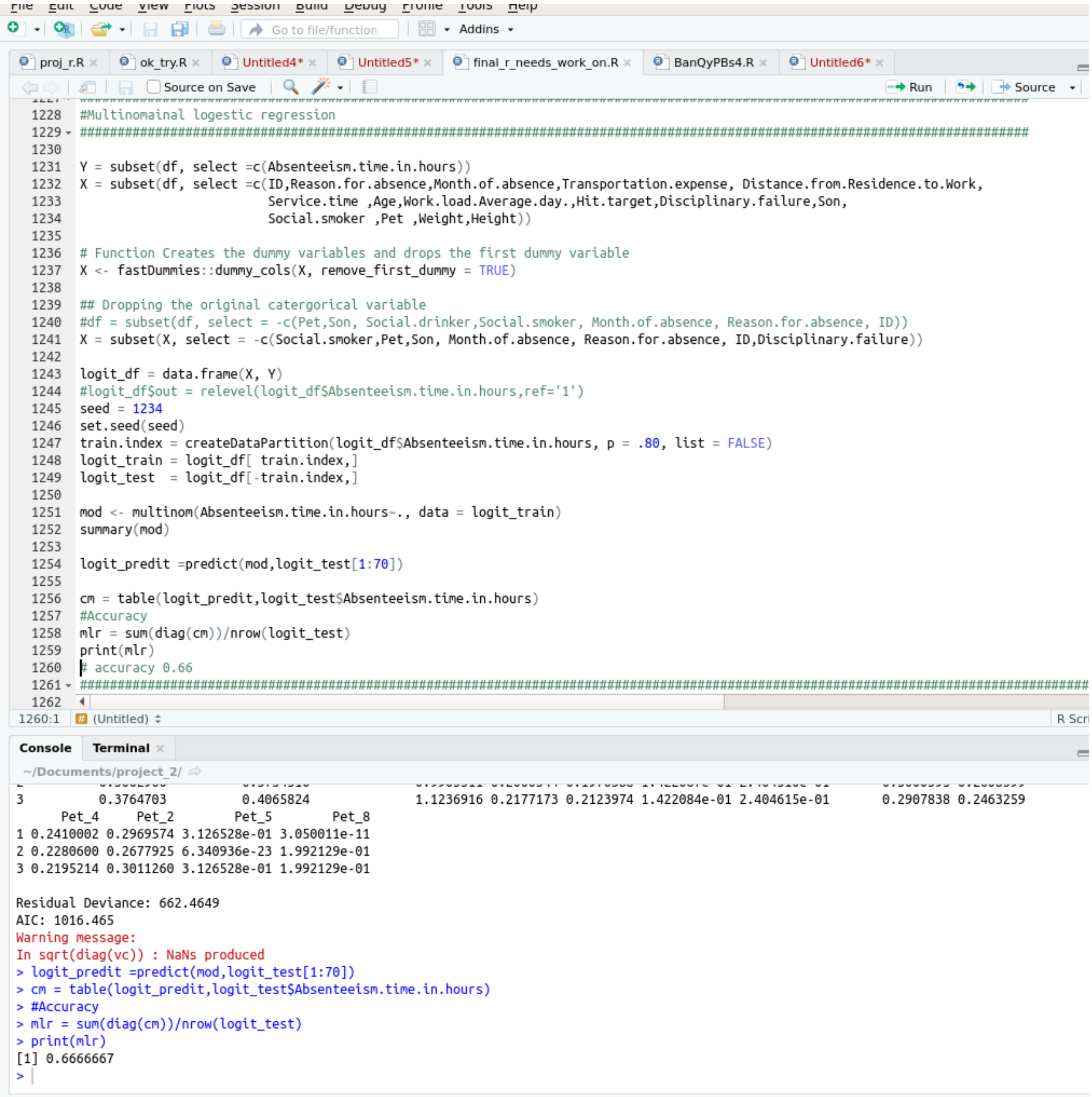


Figure A.5: Logestic regression model in r

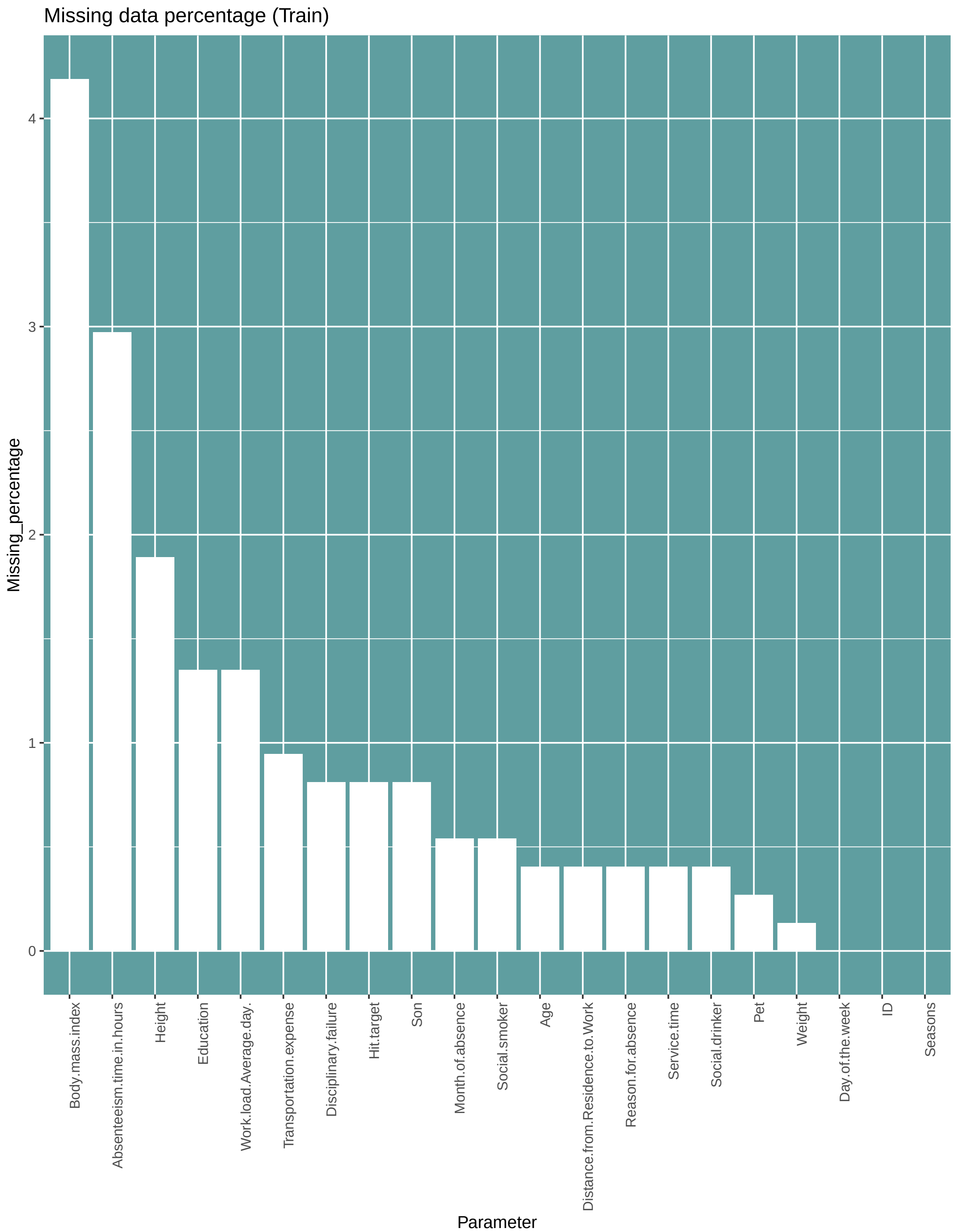


Figure A.6: missing value plot.