Employee Absenteeism

* CH SAI SASHANK

**Contents**

1 Introduction

1.1 Problem Statement

1.2 Data

2 Methodology

2.1 Exploratory Data analysis

2.2 Pre Processing

2.3 Missing-values Analysis

2.4 Outlier Analysis

2.5 Feature Selection

2.6 Feature Scaling

2.7 Dimensionality reduction

3 Modeling

3.1 Model Evaluation

3.2 Model Selection

3.3 Answers of questionnaire

Appendix A –Python code

**Chapter 1**

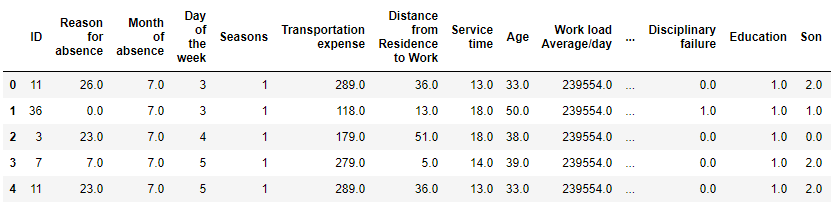
**Introduction**

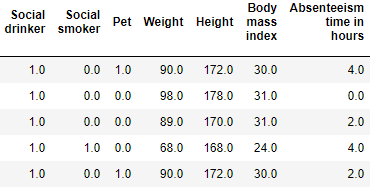
* 1. **Problem Statement**

The objective of this project is to predict the Absenteeism at work and propose any changes to make to reduce the Absenteeism by using health, residential and work data.

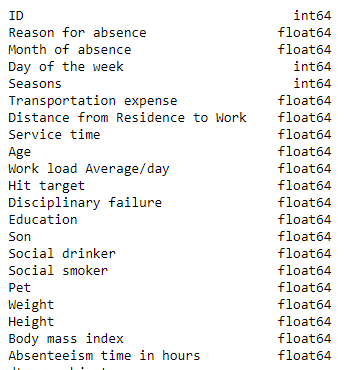
* 1. **Data Understanding**

There are 21 variables in our data in which 20 are independent variables and 1 (Absenteeism time in hours) is dependent variable. Since our target variable is continuous in nature, this is a regression problem.





As you can see in the table below we have the following variables and their data types, using which we have to correctly predict the absenteeism time in hours



**Chapter 2**

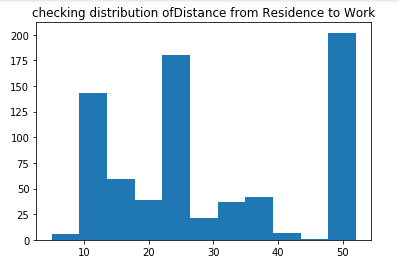
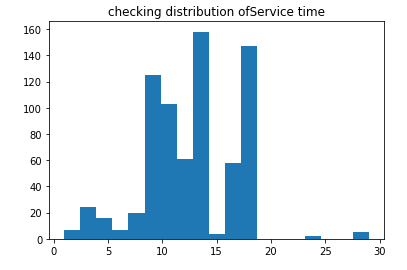
**Methodology**

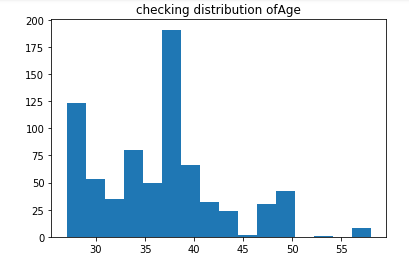
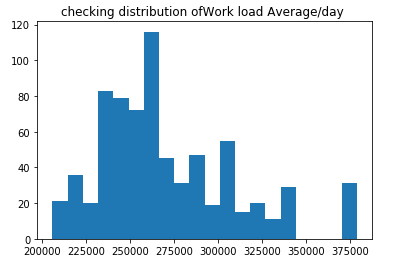
**2.1 Exploratory Data Analysis**

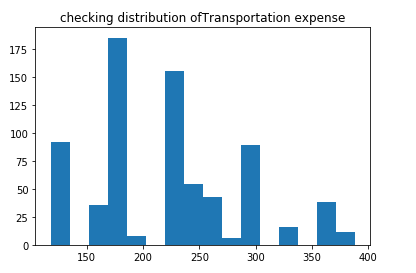
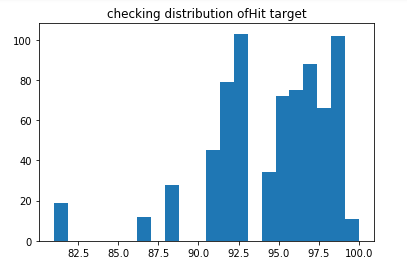
Exploratory data analysis (EDA) is an approach to analyse [data sets](https://en.wikipedia.org/wiki/Data_set)  so as to discover hidden patterns, to spot anomalies and to summarize their main characteristics, often with visual methods and graphical representations

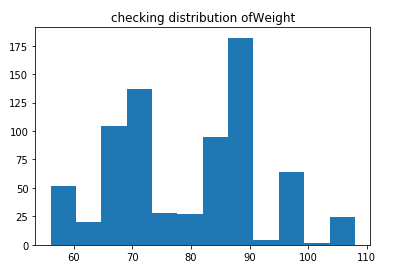
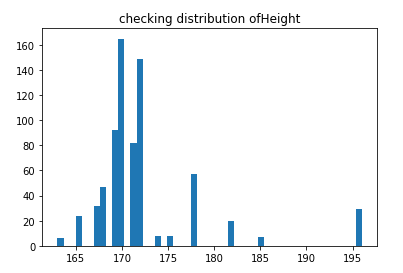
## Univariate Analysis

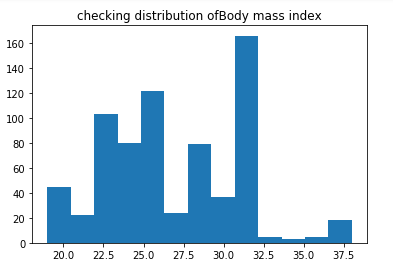
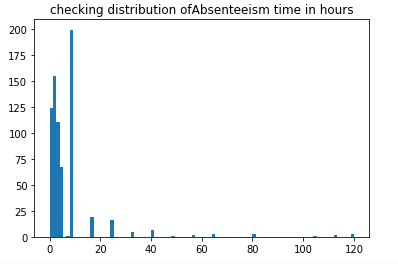
In this section, we will do univariate analysis. It is the simplest form of analysing data where we examine each variable individually. For categorical features we can use frequency table or bar plots which will calculate the number of each category in a particular variable. For numerical features, probability density plots can be used to look at the distribution of the variable.

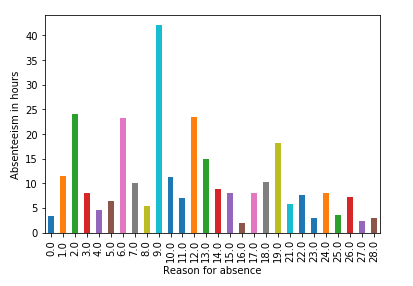
 

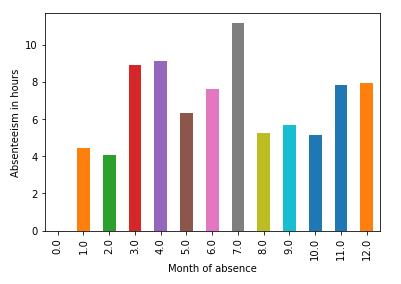
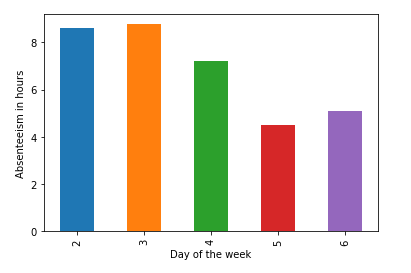
 

I have plotted histograms to check distribution of continuous variables. From above plots we can see that variables don’t have quite normal distribution. So there is need of feature scaling before feeding the data to the model.

I have used bar plots to check the distribution of categorical variables. From below Reason for absence plots we can see mean of the Absenteeism is high for **Diseases of the circulatory system.**

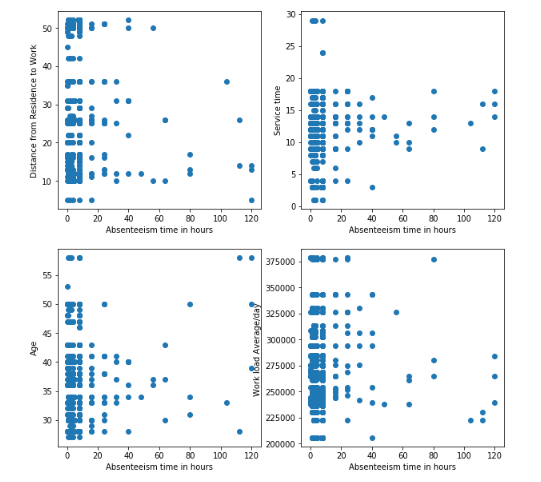


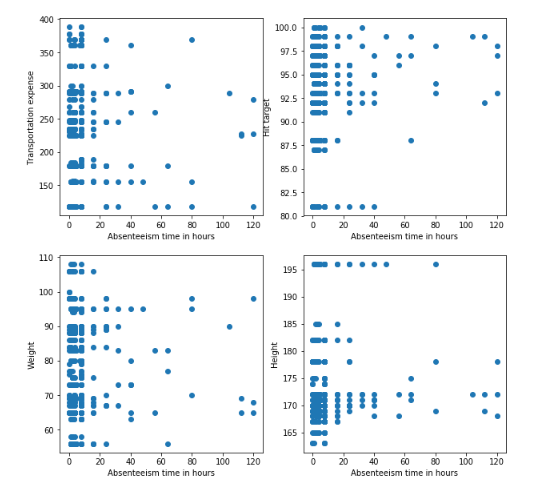
From below plots we could see that month of absence is high in 7th month and more people tend to absent on 3rd day of the week

## Bivariate Analysis

In Bivariate Analysis we will check the relation between any two variables using graphs or statistical techniques. Here I have analysed relation between target variable and other predictor variables. As target variable is continuous I have used scatter plots to check the relation between target variable and continuous variables.





From above plots we don’t see any particular pattern. There is no linear relationship between independent variables and target. There are many outliers in target variable.

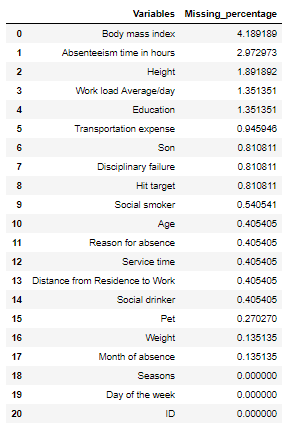
**2.1 Pre-processing**

The most important thing required is need to look at the data before we do modelling. The data must be tidy and clean to proceed further. The data must be free from noisy data known as outliers and missing values need to be handled properly. This process is known as exploratory data analysis (EDA).The pre-requisites of many machine learning algorithm is that data must be normally distributed.

In Exploratory data analysis we found that data is not normally distributed. So we must normalize the data by using normalization technique.

**2.2** **Missing values Analysis**

In statistics, *missing data*, or *missing values*, occur when no *data value* is stored for the variable in an observation. *Missing data* are a common occurrence and can have a significant effect on the conclusions that can be drawn from the *data*. If a columns has more than 30% of data as missing value either we ignore the entire column or we ignore those observations. In the given data the maximum percentage of missing value is 4.189% for **body mass index** column. So we will compute missing value for all the columns.



From missing value analysis we can see there are some variables with missing values. Percentage of missing values is not more than 30% so it is better to impute the missing values rather removing variable. I have checked the three methods median, mean and KNN imputation to impute missing values and checked which one is doing best. KNN imputation outperforms here so I have imputed missing values using KNN imputation with k=3.

**2.4 Outlier Analysis**

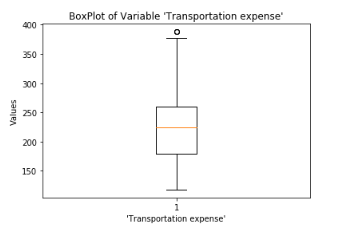
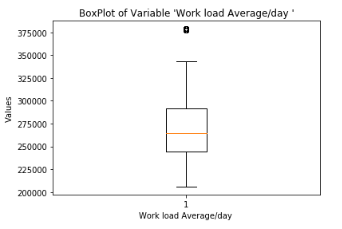
We can clearly observe from the variable distributions that most of the variables are skewed. The skew in these distributions can be most likely explained by the presence of outliers and extreme values in the data. One of the other steps of pre-processing apart from checking for normality is the presence of outliers. In this case we use a classic approach of removing outliers. We visualize the outliers using boxplots.

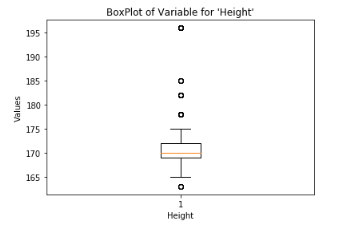
With the help of inter-quartile range, we can either replace or delete the outliers by setting the benchmark

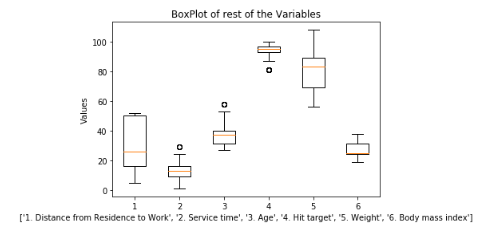
min=Q1-(1.5 \* IQR)  
max=Q3+(1.5 \* IQR)

IQR=Q3-Q1

The values which are below minimum and above maximum can either be deleted or replaced with mean values. Also can be imputed with knn imputation



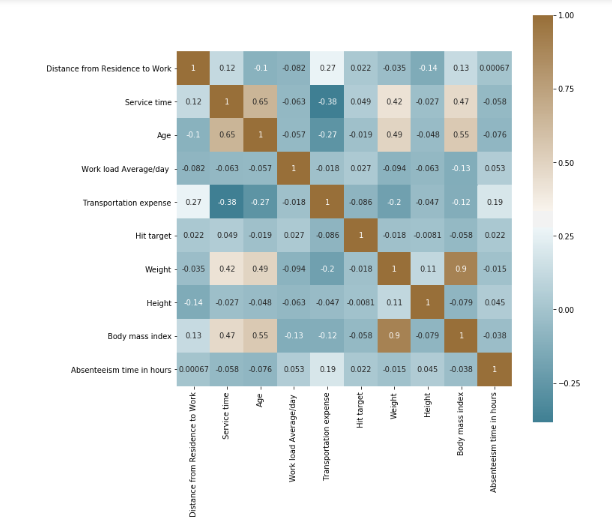


From the boxplot almost all the variables except “Distance from residence to work”, “Weight” and “Body mass index” consists of outliers.

From outlier analysis we can see there are many outliers in data so I decided not to remove outliers as we may lose useful information. I have imputed NA’s in outliers and used KNN imputation to impute those missing values.

**2.5 Feature Selection**

Before performing any type of modeling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. Selecting subset of relevant columns for the model construction is known as Feature Selection. We cannot use all the features because some features may be carrying the same information or irrelevant information which can increase overhead. To reduce overhead we adopt feature selection technique to extract meaningful features out of data. This in turn helps us to avoid the problem of multi collinearity. In this project we have selected **Correlation Analysis** for numerical variable and **ANOVA (Analysis of variance)** for categorical variable.



From correlation analysis we have found that Weight and Body mass index has high correlation (>0.7), so we have excluded the Weight column.

**2.5 Feature Scaling**

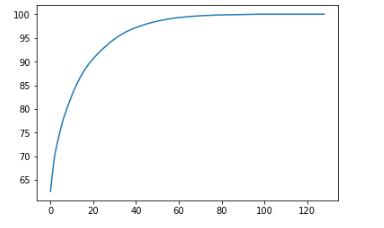
Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data pre-processing step. Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance. Since our data is not uniformly distributed we will use Normalization as Feature Scaling Method.

**2.6 Dimensionality Reduction**

Dimensionality reduction is the process of reducing the dimensionality of the feature space with consideration by obtaining a set of principal features. Dimensionality reduction can be further broken into feature selection and feature extraction. This approach can also derive informative and non-redundant features.

In this project I have used principal component analysis to extract the significant variables before building the model.

**Principal component analysis** is a method of extracting important variables (in form of components) from a large set of variables available in a data set. It extracts low dimensional set of features from a high dimensional data set with a motive to capture as much information as possible. With fewer variables, visualization also becomes much more meaningful. PCA is more useful when dealing with 3 or higher dimensional data. After creating dummy variable of categorical variables the shape of our data became 129 columns and 714 observations, this high number of columns leads to bad accuracy.



We have applied PCA algorithm on our data and from the above graph we have concluded that 45 variables out of 129 explains more than 95% of data. So we have selected only those 45 variables to feed our models.

**Chapter 3**

**Modeling**

After a thorough pre-processing we will be using some regression models on our processed data to predict the target variable. Following are the models which we have built below models

**Decision Tree**

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Each branch connects nodes with “and” and multiple branches are connected by “or”. It can be used for classification and regression. It is a supervised machine learning algorithm. Accept continuous and categorical variables as independent variables. Extremely easy to understand by the business users. Split of decision tree is seen in the below tree. The RMSE value and R^2 value for our project in R and Python are –

|  |  |
| --- | --- |
| **Decision Tree** | **PYTHON** |
| **RMSE Train** | 0.577 |
| **RMSE Test** | 0.548 |
| **R^2 Test** | 0.97 |

**Random Forest**

Random Forest is an ensemble technique that consists of many decision trees. The idea behind Random Forest is to build n number of trees to have more accuracy in dataset. It is called random forest as we are building n no. of trees randomly. In other words, to build the decision trees it selects randomly n no of variables and n no of observations to build each decision tree. It means to build each decision tree on random forest we are not going to use the same data. The RMSE value and R^2 value for our project in R and Python are –

|  |  |
| --- | --- |
| **Random Forest** | **PYTHON** |
| **RMSE Train** | 0.034 |
| **RMSE Test** | 0.031 |
| **R^2 Test** | 0.99 |

**Linear Regression**

Linear Regression is one of the statistical methods of prediction. It is applicable only on continuous data. To build any model we have some assumptions to put on data and model. Here are the assumptions to the linear regression model.

|  |  |
| --- | --- |
| **Linear Regression** | **PYTHON** |
| **RMSE Train** | 2.713e-15 |
| **RMSE Test** | 0.0017 |
| **R^2 Test** | 1 |

**Gradient boosting**

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

|  |  |
| --- | --- |
| **Gradient boosting** | **PYTHON** |
| **RMSE Train** | 0.001 |
| **RMSE Test** | 0.009 |
| **R^2** | 0.99 |

**3.1 Model Evaluation**

In the previous chapter we have seen the Root Mean Square Error (RMSE) and R-Squared Value of different models. Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are, RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Whereas R-squared is a relative measure of fit, RMSE is an absolute measure of fit. As the square root of a variance, RMSE can be interpreted as the standard deviation of the unexplained variance, and has the useful property of being in the same units as the response variable. Lower values of RMSE and higher value of R-Squared Value indicate better fit.

**Conclusion**

**3.2 Model Selection**

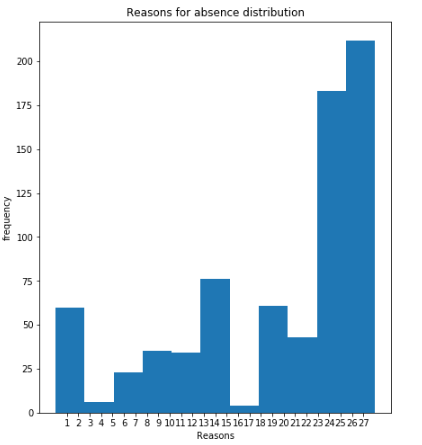
From the observation of all RMSE Value and R-Squared Value we have concluded that gradient boosting Model has minimum value of RMSE and it’s R-Squared Value is also nearly 1.

The RMSE value of testing data and Training does not differs a lot this implies that it is not the case of overfitting.

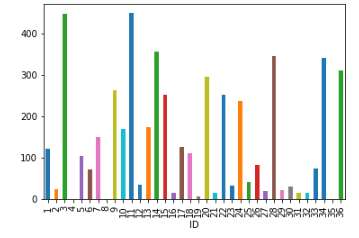
**3.3 Answers of Questionnaire**

**The Changes which company should bring to reduce the number of absenteeism –**

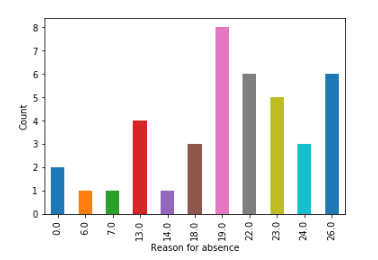
* Most often Reason for absence are dental consultation and physiotherapy, So Company should take care of it.



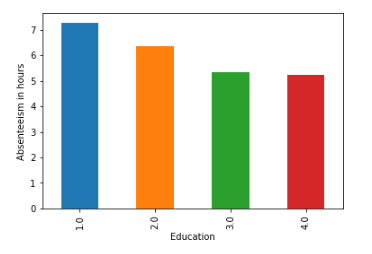
* Some employee with **ID 3, 11, 14** are often absent from work, company should take action against them.



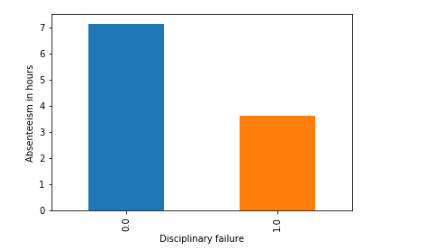
* Reasons for absence of highest absentee (ID=11) are injury, poisoning and certain other consequences of external causes. So company must concentrate on health concerns.



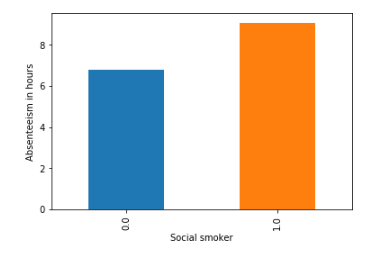
* It is observed that employee with low education have maximum absentee time.



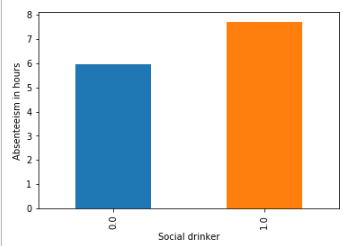
* People who are claiming as disciplinary failure have maximum absentee time.



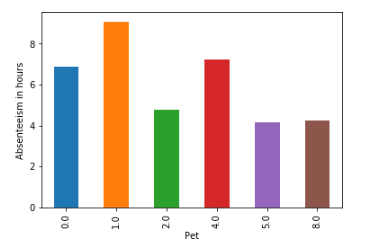
* Employees who are social smoker have more absentee hour than non- social smoker.



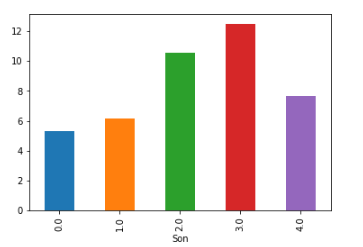
* Employees who are social drinker have more absentee hour than non- social drinker



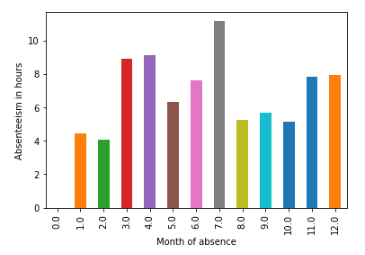
* Employee with 1 pet tends to have more absentee time.



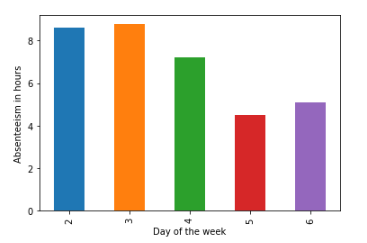
* Employee who is having 3 sons have more absentee time.



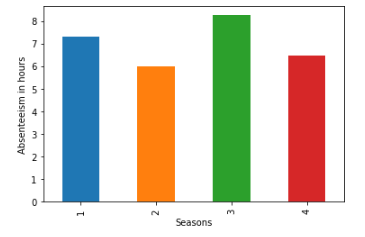
* It is observed that employees are having more absentee time in 7th month of year (July).



* It is observed that employees are having more absentee time in 3rd day of week (Tuesday).



* It is observed that employees are having more absentee time in season 3(Winter)



**Appendix –Python code**

#loading libraries

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from fancyimpute import KNN

from scipy import stats

from sklearn.metrics import r2\_score

%matplotlib inline

#Setting working directory

os.chdir("D:\Data Science edwisor\Projects\Employee Absentism")

os.getcwd()

#loading data

df=pd.read\_excel('Absenteeism\_at\_work\_Project.xls')

# First 5 rows of data

df.head()

# Data Types of all the variables

df.dtypes

# Number of Unique values present in each variable

df.nunique()

#dimensions of the data

df.shape

# categorising the variables

continuous\_vars = ['Distance from Residence to Work', 'Service time', 'Age', 'Work load Average/day ', 'Transportation expense',

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

categorical\_vars = ['ID','Reason for absence','Month of absence','Day of the week',

'Seasons','Disciplinary failure', 'Education', 'Social drinker',

'Social smoker', 'Pet', 'Son']

#checking distribution of continuous variables

for i in continuous\_vars :

plt.hist(df[i].dropna(),bins='auto')

plt.title('checking distribution of' + str(i))

plt.show()

#checking the mean of Absenteeism time in hours

df['Absenteeism time in hours'].dropna().mean()

#checking reason for absenteeism

plt.figure(figsize=(7,8))

plt.hist(df['Reason for absence'].dropna(),bins='auto')

plt.xticks(range(1,28))

plt.xlabel('Reasons')

plt.ylabel('frequency')

plt.title('Reasons for absence distribution')

plt.show()

# could see people with dental consultation and physiotherapy tend to be absent more often.

#checking education variable

plt.hist(df['Education'].dropna(),bins=4)

plt.xticks(range(1,4))

plt.xlabel('Education')

plt.ylabel('Count')

plt.title('Education distribution')

plt.show()

## grouping the data using ID and plotting bar plot

df.groupby('ID').sum()['Absenteeism time in hours'].plot.bar()

#checking what are reasons for absence of highest absentee (ID=11)

df.iloc[np.where(df['ID']==11)].groupby('Reason for absence').count()['ID'].plot.bar()

plt.ylabel('Count')

## grouping the data using Reason for absence and plotting bar plot

df.groupby('Reason for absence').mean()['Absenteeism time in hours'].plot.bar()

plt.ylabel('Absenteeism in hours')

df.groupby(['Month of absence']).mean()['Absenteeism time in hours'].plot.bar()

plt.ylabel('Absenteeism in hours')

df.groupby(['Day of the week']).mean()['Absenteeism time in hours'].plot.bar()

plt.ylabel('Absenteeism in hours')

df.groupby(['Seasons']).mean()['Absenteeism time in hours'].plot.bar()

plt.ylabel('Absenteeism in hours')

df.groupby([categorical\_vars[5]]).mean()['Absenteeism time in hours'].plot.bar()

plt.ylabel('Absenteeism in hours')

df.groupby([categorical\_vars[6]]).mean()['Absenteeism time in hours'].plot.bar()

plt.ylabel('Absenteeism in hours')

df.groupby([categorical\_vars[7]]).mean()['Absenteeism time in hours'].plot.bar()

plt.ylabel('Absenteeism in hours')

df.groupby([categorical\_vars[8]]).mean()['Absenteeism time in hours'].plot.bar()

plt.ylabel('Absenteeism in hours')

df.groupby([categorical\_vars[9]]).mean()['Absenteeism time in hours'].plot.bar()

plt.ylabel('Absenteeism in hours')

df.groupby([categorical\_vars[10]]).mean()['Absenteeism time in hours'].plot.bar()

## plotting scatter plot for target variable vs each continuous variable

fig,axis = plt.subplots(nrows=2,ncols=2,figsize=(10,10))

axis[0,0].scatter(df['Absenteeism time in hours'],df[continuous\_vars[0]])

axis[0,0].set\_xlabel('Absenteeism time in hours')

axis[0,0].set\_ylabel(continuous\_vars[0])

axis[0,1].scatter(df['Absenteeism time in hours'],df[continuous\_vars[1]])

axis[0,1].set\_xlabel('Absenteeism time in hours')

axis[0,1].set\_ylabel(continuous\_vars[1])

axis[1,0].scatter(df['Absenteeism time in hours'],df[continuous\_vars[2]])

axis[1,0].set\_xlabel('Absenteeism time in hours')

axis[1,0].set\_ylabel(continuous\_vars[2])

axis[1,1].scatter(df['Absenteeism time in hours'],df[continuous\_vars[3]])

axis[1,1].set\_xlabel('Absenteeism time in hours')

axis[1,1].set\_ylabel(continuous\_vars[3])

## plotting scatter plot for target variable vs each continuous variable

fig,axis = plt.subplots(nrows=2,ncols=2,figsize=(10,10))

axis[0,0].scatter(df['Absenteeism time in hours'],df[continuous\_vars[4]])

axis[0,0].set\_xlabel('Absenteeism time in hours')

axis[0,0].set\_ylabel(continuous\_vars[4])

axis[0,1].scatter(df['Absenteeism time in hours'],df[continuous\_vars[5]])

axis[0,1].set\_xlabel('Absenteeism time in hours')

axis[0,1].set\_ylabel(continuous\_vars[5])

axis[1,0].scatter(df['Absenteeism time in hours'],df[continuous\_vars[6]])

axis[1,0].set\_xlabel('Absenteeism time in hours')

axis[1,0].set\_ylabel(continuous\_vars[6])

axis[1,1].scatter(df['Absenteeism time in hours'],df[continuous\_vars[7]])

axis[1,1].set\_xlabel('Absenteeism time in hours')

axis[1,1].set\_ylabel(continuous\_vars[7])

#Creating dataframe with missing values present in each variable

missing\_val = pd.DataFrame(df.isnull().sum()).reset\_index()

#Renaming variables of missing\_val dataframe

missing\_val = missing\_val.rename(columns = {'index': 'Variables', 0: 'Missing\_percentage'})

#Calculating percentage missing value

missing\_val['Missing\_percentage'] = (missing\_val['Missing\_percentage']/len(df))\*100

# Sorting missing\_val in Descending order

missing\_val = missing\_val.sort\_values('Missing\_percentage', ascending = False).reset\_index(drop = True)

missing\_val

# Droping observation in which "Absenteeism time in hours" has missing value

df = df.drop(df[df['Absenteeism time in hours'].isnull()].index, axis=0)

print(df.shape)

print(df['Absenteeism time in hours'].isnull().sum())

df['Body mass index'].iloc[12]

#create missing value

df['Body mass index'].iloc[12] = np.nan

#Apply KNN imputation algorithm

df = pd.DataFrame(KNN(k = 3).fit\_transform(df), columns = df.columns)

df['Body mass index'].iloc[12]

# Checking if all the missing value imputed

df.isnull().sum().sum()

# Ploting BoxPlot of continuous variables

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

plt.xlabel("'Transportation expense'")

plt.title("BoxPlot of Variable 'Transportation expense'")

plt.ylabel('Values')

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

plt.xlabel("Work load Average/day ")

plt.title("BoxPlot of Variable 'Work load Average/day '")

plt.ylabel('Values')

plt.boxplot(df['Height'])

plt.xlabel("Height")

plt.title("BoxPlot of Variable for 'Height'")

plt.ylabel('Values')

plt.boxplot([ df['Distance from Residence to Work'], df['Service time'], df['Age'], df['Hit target'], df['Weight'], df['Body mass index']])

plt.xlabel(['1. Distance from Residence to Work', '2. Service time', '3. Age', '4. Hit target', '5. Weight', '6. Body mass index'])

plt.title("BoxPlot of rest of the Variables")

plt.ylabel('Values')

# list of variables which doesn't have outlier

neglect = ['Distance from Residence to Work', 'Weight', 'Body mass index']

# Looping over all continuou variables to detect and remove Outliers

for i in continuous\_vars:

# Avoiding the variables which doesn't have outlier

if i in neglect:

continue

# Getting 75 and 25 percentile of variable "i"

q75, q25 = np.percentile(df[i], [75,25])

# Calculating Interquartile range

iqr = q75 - q25

# Calculating upper extream and lower extream

minimum = q25 - (iqr\*1.5)

maximum = q75 + (iqr\*1.5)

# Replacing all the outliers value to NA

df.loc[df[i]< minimum,i] = np.nan

df.loc[df[i]> maximum,i] = np.nan

# Imputing missing values with KNN

df = pd.DataFrame(KNN(k = 3).fit\_transform(df), columns = df.columns)

# Checking if there is any missing value

df.isnull().sum().sum()

##Correlation analysis for continuous variables

#Correlation plot

df\_corr = df.loc[:,continuous\_vars]

#Set the width and hieght of the plot

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

#Generate correlation matrix

corr = df\_corr.corr()

#Plot using seaborn library

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool),

cmap=sns.diverging\_palette(220, 50, as\_cmap=True),

square=True, ax=ax, annot = True)

plt.plot()

#loop for ANOVA test Since the target variable is continuous

for i in categorical\_vars:

f, p = stats.f\_oneway(df[i], df["Absenteeism time in hours"])

print("P value for variable "+str(i)+" is "+str(p))

# Droping the variables which has redundant information

to\_drop = ['Weight']

df = df.drop(to\_drop, axis = 1)

# Updating the Continuous Variables and Categorical Variables after droping some variables

continuous\_vars = [i for i in continuous\_vars if i not in to\_drop]

categorical\_vars = [i for i in categorical\_vars if i not in to\_drop]

clean\_data = df.copy()

# Checking if there is any normally distributed variable in data

for i in continuous\_vars:

if i == 'Absenteeism time in hours':

continue

sns.distplot(df[i],bins = 'auto')

plt.title("Checking Distribution for Variable "+str(i))

plt.ylabel("Density")

plt.show()

# Since there is no normally distributed curve we will use Normalizationg for Feature Scalling

# #Normalization

for i in continuous\_vars:

if i == 'Absenteeism time in hours':

continue

df[i] = (df[i] - df[i].min())/(df[i].max()-df[i].min())

# Get dummy variables for categorical variables

df = pd.get\_dummies(data = df, columns = categorical\_vars)

# Copying dataframe

df1 = df.copy()

df.iloc[:,8].head()

df.iloc[:, df.columns != 'Absenteeism time in hours'].head()

# Using train\_test\_split sampling function for test and train data split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split( df.iloc[:, df.columns != 'Absenteeism time in hours'], df.iloc[:, 8], test\_size = 0.20)

#Decision tree

# Importing libraries for Decision Tree

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

# Building model on top of training dataset

fit\_DT = DecisionTreeRegressor(max\_depth = 2).fit(X\_train,y\_train)

# Calculating RMSE for training data to check for over fitting

pred\_train = fit\_DT.predict(X\_train)

rmse\_for\_train = np.sqrt(mean\_squared\_error(y\_train,pred\_train))

# Calculating RMSE for test data to check accuracy

pred\_test = fit\_DT.predict(X\_test)

rmse\_for\_test =np.sqrt(mean\_squared\_error(y\_test,pred\_test))

print("Root Mean Squared Error For Training data = "+str(rmse\_for\_train))

print("Root Mean Squared Error For Test data = "+str(rmse\_for\_test))

print("R^2 Score(coefficient of determination) = "+str(r2\_score(y\_test,pred\_test)))

#Random forest

# Importing libraries for Random Forest

from sklearn.ensemble import RandomForestRegressor

# Building model on top of training dataset

fit\_RF = RandomForestRegressor(n\_estimators = 500).fit(X\_train,y\_train)

# Calculating RMSE for training data to check for over fitting

pred\_train = fit\_RF.predict(X\_train)

rmse\_for\_train = np.sqrt(mean\_squared\_error(y\_train,pred\_train))

# Calculating RMSE for test data to check accuracy

pred\_test = fit\_RF.predict(X\_test)

rmse\_for\_test =np.sqrt(mean\_squared\_error(y\_test,pred\_test))

print("Root Mean Squared Error For Training data = "+str(rmse\_for\_train))

print("Root Mean Squared Error For Test data = "+str(rmse\_for\_test))

print("R^2 Score(coefficient of determination) = "+str(r2\_score(y\_test,pred\_test)))

#Linear regression

# Importing libraries for Linear Regression

from sklearn.linear\_model import LinearRegression

# Building model on top of training dataset

fit\_LR = LinearRegression().fit(X\_train , y\_train)

# Calculating RMSE for training data to check for over fitting

pred\_train = fit\_LR.predict(X\_train)

rmse\_for\_train = np.sqrt(mean\_squared\_error(y\_train,pred\_train))

# Calculating RMSE for test data to check accuracy

pred\_test = fit\_LR.predict(X\_test)

rmse\_for\_test =np.sqrt(mean\_squared\_error(y\_test,pred\_test))

print("Root Mean Squared Error For Training data = "+str(rmse\_for\_train))

print("Root Mean Squared Error For Test data = "+str(rmse\_for\_test))

print("R^2 Score(coefficient of determination) = "+str(r2\_score(y\_test,pred\_test)))

#Gradient boosting

# Importing library for GradientBoosting

from sklearn.ensemble import GradientBoostingRegressor

# Building model on top of training dataset

fit\_GB = GradientBoostingRegressor().fit(X\_train, y\_train)

# Calculating RMSE for training data to check for over fitting

pred\_train = fit\_GB.predict(X\_train)

rmse\_for\_train = np.sqrt(mean\_squared\_error(y\_train,pred\_train))

# Calculating RMSE for test data to check accuracy

pred\_test = fit\_GB.predict(X\_test)

rmse\_for\_test =np.sqrt(mean\_squared\_error(y\_test,pred\_test))

print("Root Mean Squared Error For Training data = "+str(rmse\_for\_train))

print("Root Mean Squared Error For Test data = "+str(rmse\_for\_test))

print("R^2 Score(coefficient of determination) = "+str(r2\_score(y\_test,pred\_test)))

target = df['Absenteeism time in hours']

df.drop(['Absenteeism time in hours'], inplace = True, axis=1)

df.shape

from sklearn.decomposition import PCA

# Converting data to numpy array

X = df1.values

# Data has 129 variables so no of components of PCA = 129

pca = PCA(n\_components=129)

pca.fit(X)

# The amount of variance that each PC explains

var= pca.explained\_variance\_ratio\_

# Cumulative Variance explains

var1=np.cumsum(np.round(pca.explained\_variance\_ratio\_, decimals=4)\*100)

plt.plot(var1)

plt.show()

# From the above plot selecting 45 components since it explains almost 95+ % data variance

pca = PCA(n\_components=45)

# Fitting the selected components to the data

pca.fit(X)

# Using train\_test\_split sampling function for test and train data split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,target, test\_size=0.2)

#Decision tree

# Importing libraries for Decision Tree

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

# Building model on top of training dataset

fit\_DT = DecisionTreeRegressor(max\_depth = 2).fit(X\_train,y\_train)

# Calculating RMSE for training data to check for over fitting

pred\_train = fit\_DT.predict(X\_train)

rmse\_for\_train = np.sqrt(mean\_squared\_error(y\_train,pred\_train))

# Calculating RMSE for test data to check accuracy

pred\_test = fit\_DT.predict(X\_test)

rmse\_for\_test =np.sqrt(mean\_squared\_error(y\_test,pred\_test))

print("Root Mean Squared Error For Training data = "+str(rmse\_for\_train))

print("Root Mean Squared Error For Test data = "+str(rmse\_for\_test))

print("R^2 Score(coefficient of determination) = "+str(r2\_score(y\_test,pred\_test)))

#Random forest

# Importing libraries for Random Forest

from sklearn.ensemble import RandomForestRegressor

# Building model on top of training dataset

fit\_RF = RandomForestRegressor(n\_estimators = 500).fit(X\_train,y\_train)

# Calculating RMSE for training data to check for over fitting

pred\_train = fit\_RF.predict(X\_train)

rmse\_for\_train = np.sqrt(mean\_squared\_error(y\_train,pred\_train))

# Calculating RMSE for test data to check accuracy

pred\_test = fit\_RF.predict(X\_test)

rmse\_for\_test =np.sqrt(mean\_squared\_error(y\_test,pred\_test))

print("Root Mean Squared Error For Training data = "+str(rmse\_for\_train))

print("Root Mean Squared Error For Test data = "+str(rmse\_for\_test))

print("R^2 Score(coefficient of determination) = "+str(r2\_score(y\_test,pred\_test)))

#Linear regression

# Importing libraries for Linear Regression

from sklearn.linear\_model import LinearRegression

# Building model on top of training dataset

fit\_LR = LinearRegression().fit(X\_train , y\_train)

# Calculating RMSE for training data to check for over fitting

pred\_train = fit\_LR.predict(X\_train)

rmse\_for\_train = np.sqrt(mean\_squared\_error(y\_train,pred\_train))

# Calculating RMSE for test data to check accuracy

pred\_test = fit\_LR.predict(X\_test)

rmse\_for\_test =np.sqrt(mean\_squared\_error(y\_test,pred\_test))

print("Root Mean Squared Error For Training data = "+str(rmse\_for\_train))

print("Root Mean Squared Error For Test data = "+str(rmse\_for\_test))

print("R^2 Score(coefficient of determination) = "+str(r2\_score(y\_test,pred\_test)))

# Importing library for Gradient Boosting

from sklearn.ensemble import GradientBoostingRegressor

# Building model on top of training dataset

fit\_GB = GradientBoostingRegressor().fit(X\_train, y\_train)

# Calculating RMSE for training data to check for over fitting

pred\_train = fit\_GB.predict(X\_train)

rmse\_for\_train = np.sqrt(mean\_squared\_error(y\_train,pred\_train))

# Calculating RMSE for test data to check accuracy

pred\_test = fit\_GB.predict(X\_test)

rmse\_for\_test =np.sqrt(mean\_squared\_error(y\_test,pred\_test))

print("Root Mean Squared Error For Training data = "+str(rmse\_for\_train))

print("Root Mean Squared Error For Test data = "+str(rmse\_for\_test))

print("R^2 Score(coefficient of determination) = "+str(r2\_score(y\_test,pred\_test)))