Project

Employee Absenteeism

Aditya Tiwari
Adi138250@gmail.com

Contents

1. Introduction

1.1 Problem Statement

XYZ is a courier company. As we appreciate that human capital plays an important role in collection, transportation and delivery. The company is passing through genuine issue of Absenteeism. The company has shared it dataset and requested to have an answer on the following areas:

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

1.2 Data

1. Individual identification (ID)
2. Reason for absence (ICD).

Absences attested by the International Code of Diseases (ICD) stratified into 21 categories (I to XXI) as follows:

I Certain infectious and parasitic diseases
II Neoplasms
III Diseases of the blood and blood-forming organs and certain disorders involving the immune mechanism
IV Endocrine, nutritional and metabolic diseases
V Mental and behavioural disorders
VI Diseases of the nervous system
VII Diseases of the eye and adnexa+ Fog
VIII Diseases of the ear and mastoid process
IX Diseases of the circulatory system
X Diseases of the respiratory system
XI Diseases of the digestive system
XII Diseases of the skin and subcutaneous tissue
XIII Diseases of the musculoskeletal system and connective tissue
XIV Diseases of the genitourinary system
XV Pregnancy, childbirth and the puerperium
XVI Certain conditions originating in the perinatal period
XVII Congenital malformations, deformations and chromosomal abnormalities
XVIII Symptoms, signs and abnormal clinical and laboratory findings, not elsewhere classified
XIX Injury, poisoning and certain other consequences of external causes
XX External causes of morbidity and mortality
XXI Factors influencing health status and contact with health services.

And 7 categories without (CID) patient follow-up (22), medical consultation (23), blood

donation (24), laboratory examination (25), unjustified absence (26), physiotherapy (27), dental consultation (28).

3. Month of absence
4. Day of the week (Monday (2), Tuesday (3), Wednesday (4), Thursday (5), Friday (6))
5. Seasons (summer (1), autumn (2), winter (3), spring (4))
6. Transportation expense
7. Distance from Residence to Work (kilometers)
8. Service time
9. Age
10. Work load Average/day
11. Hit target
12. Disciplinary failure (yes=1; no=0)
13. Education (high school (1), graduate (2), postgraduate (3), master and doctor (4))
14. Son (number of children)
15. Social drinker (yes=1; no=0)
16. Social smoker (yes=1; no=0)
17. Pet (number of pet)
18. Weight
19. Height
20. Body mass index
21. Absenteeism time in hours (target)

Table : Employee Absenteeism Sample Data (Columns: 1-8)

ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Wor	Service time
11	26	7	3	1	289	36	13
36	0	7	3	1	118	13	18

Table : Employee Absenteeism Sample Data (Columns: 9-17)

Age	Work load Average/day	Hit target	Disciplina	Education	Son	Social drinker	Social smoker	Pet
33	239554	97	0	1	2	1	0	1
50	239554	97	1	1	1	1	0	0

Table : Employee Absenteeism Sample Data (Columns: 17-21)

Weight	Height	Body mass index	Absenteeism time in hours
90	172	30	4
98	178	31	0

Below are the variables present in Employee Absenteeism dataset

Table: Employee Absenteeism

s.no	Variables
1	ID
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	Height
20	Body mass index
21	Absenteeism time in hours

2. Methodology

2.1 Pre-Processing

Any predictive modeling requires that we look at the data before we start modeling. We decided to simply remove few variables after loading data set but here we have dropped few variables after correlation test for continuous variables and anova test for categorical variables. Since our target variable is continuous and few variables are categorical hence we have applied anova test.

However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**. To start this process, we will first check the presence of missing values in data set.

2.1.1 Missing Value Analysis

Missing values Analysis is required to be done so that we can check if there is any missing data. In case data is missing at few places we will impute those missing values by different methods in order to generate appropiate results. In our case we have missing values we will proceed to impute missing values. Below table illustrate missing values present in variables of the dataset.

Sr. No.	Variables	Number of Missing Values
1	ID	0
2	Reason for absence	3
3	Month of absence	1
4	Day of the week	0
5	Seasons	0
6	Transportation expense	7
7	Distance from Residence to Work	3
8	Service time	3
9	Age	3
10	Work load Average/day	10
11	Hit target	6
12	Disciplinary failure	6
13	Education	10
14	Son	6
15	Social drinker	3
16	Social smoker	4
17	Pet	2
18	Weight	1
19	Height	14
20	Body mass index	31
21	Absenteeism time in hours	22

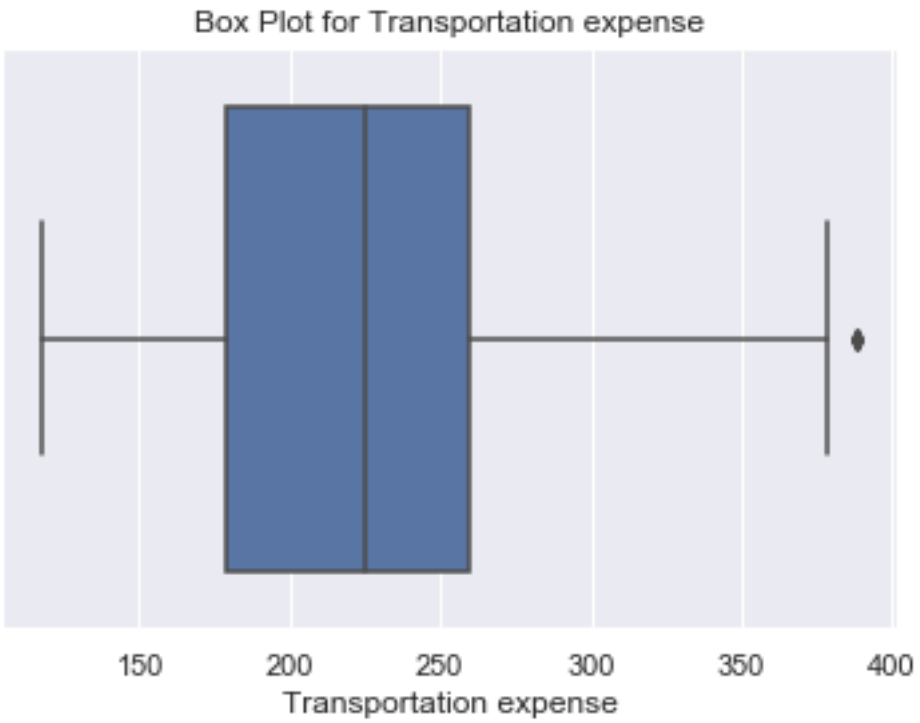
In above table we can clearly see the number of missing values present in each variable. So, we have imputed these values. We applied mean, median and Knn method to impute , where we found that knn shows best values to impute, hence we made knn method in use to replace missing values with NA and later with knn.

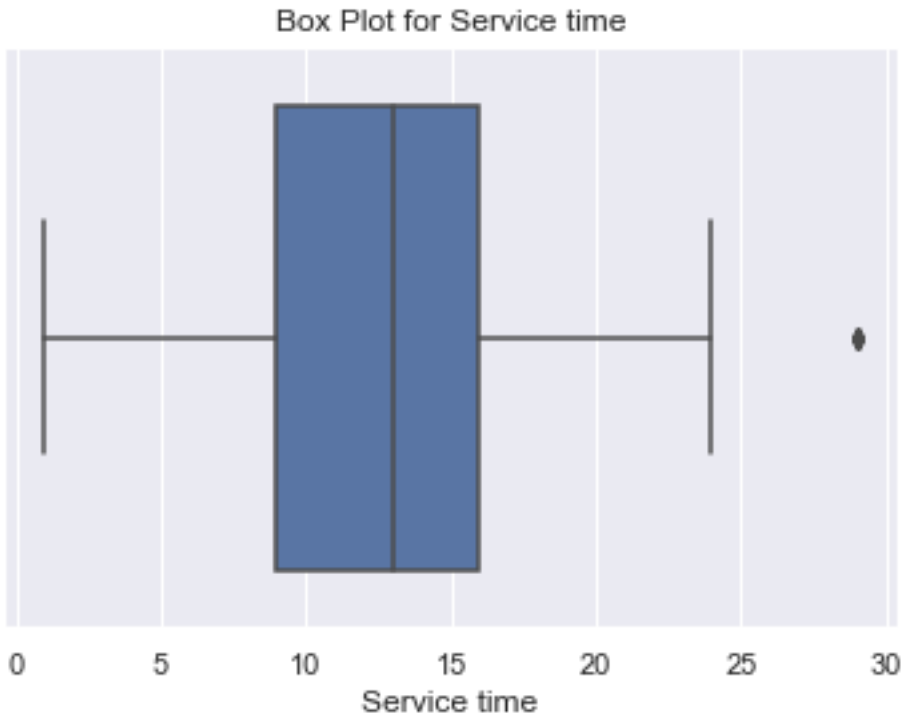
2.1.2 Outlier Analysis

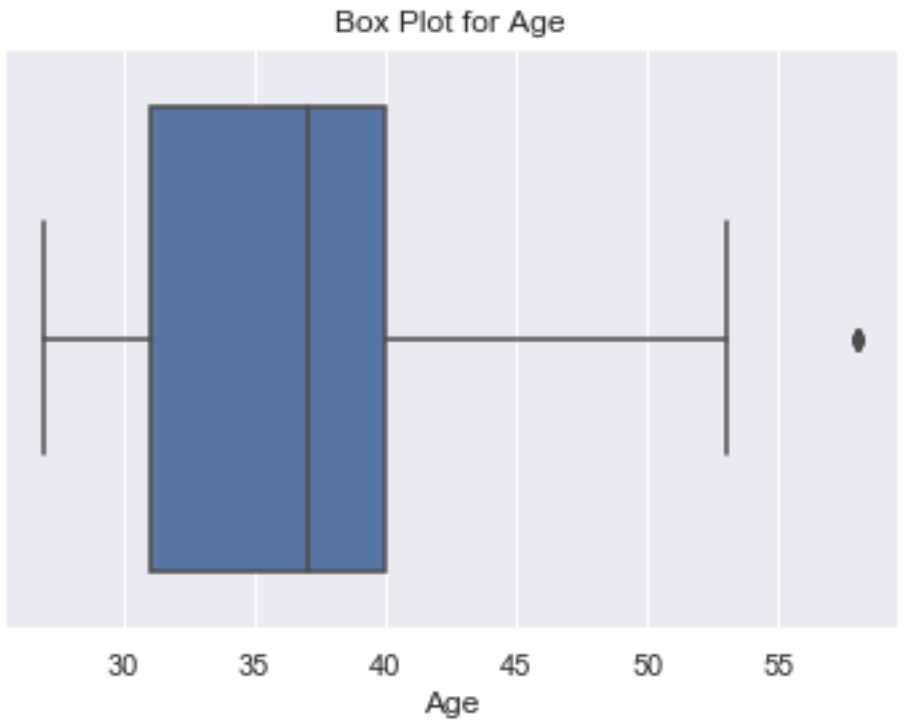
The Other steps of Preprocessing Technique is Outliers analysis, an outlier is an observation point that is distant from other observations. Outliers in data can be good and it can be bad as well. Here in our case we don't want outliers the reason for removing these outliers instead of substituting them with other balancing values (such as mean, median or knn method) because we expect them to be relatively random values and replacing them with set values may cause inaccuracy in analysis later.
The outlier analysis is done by plotting the box plot. Boxplot is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles.

Fig. 1.0

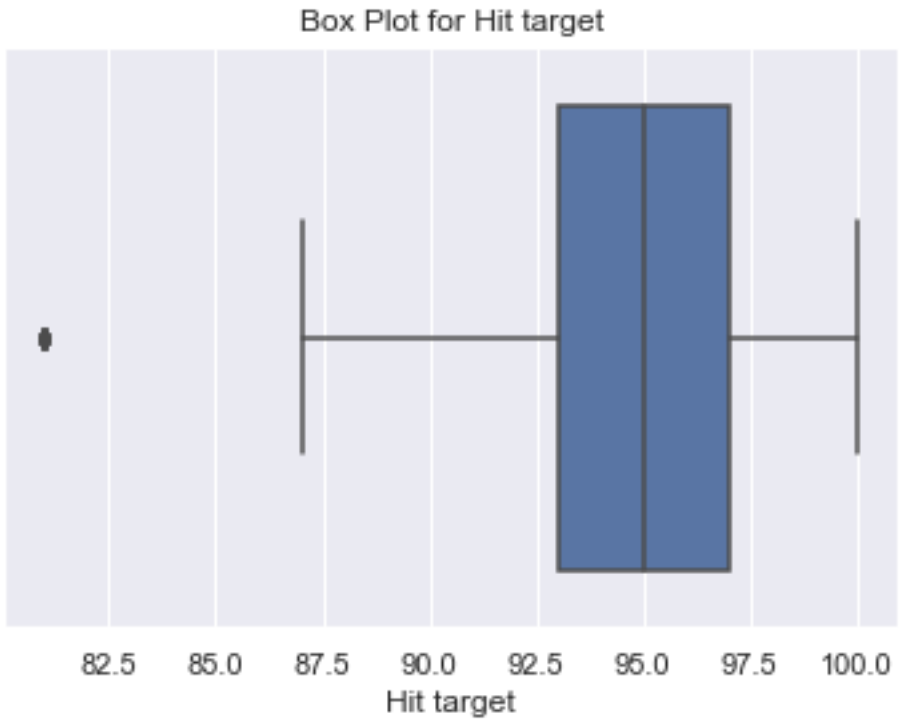
```
#Draw box plot
for i in variable_num:
    sns.boxplot(emp[i])
    plt.title("Box Plot for "+str(i))
    plt.show()
```

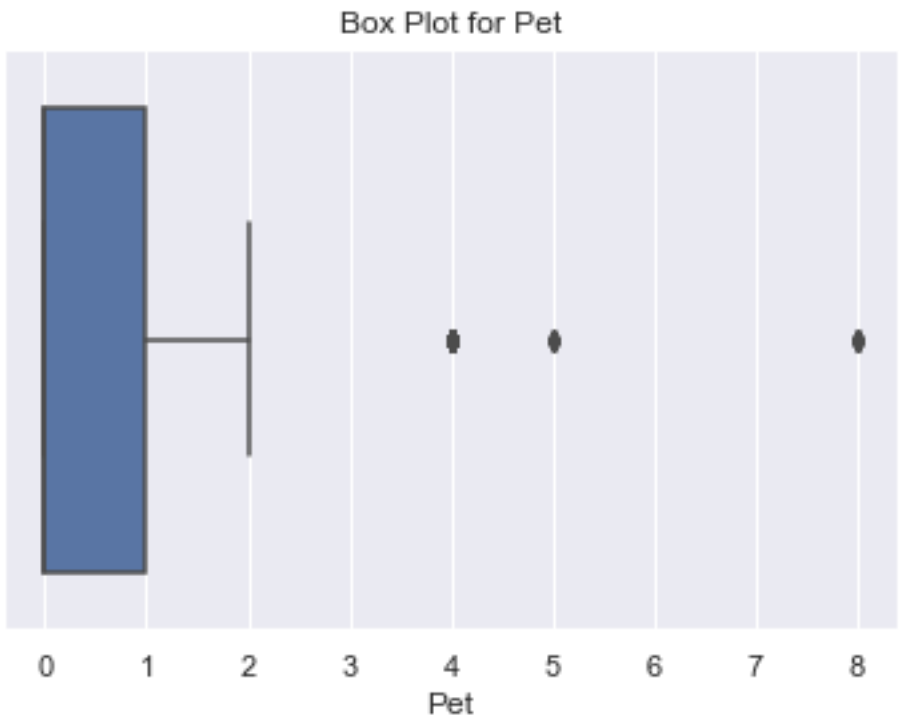


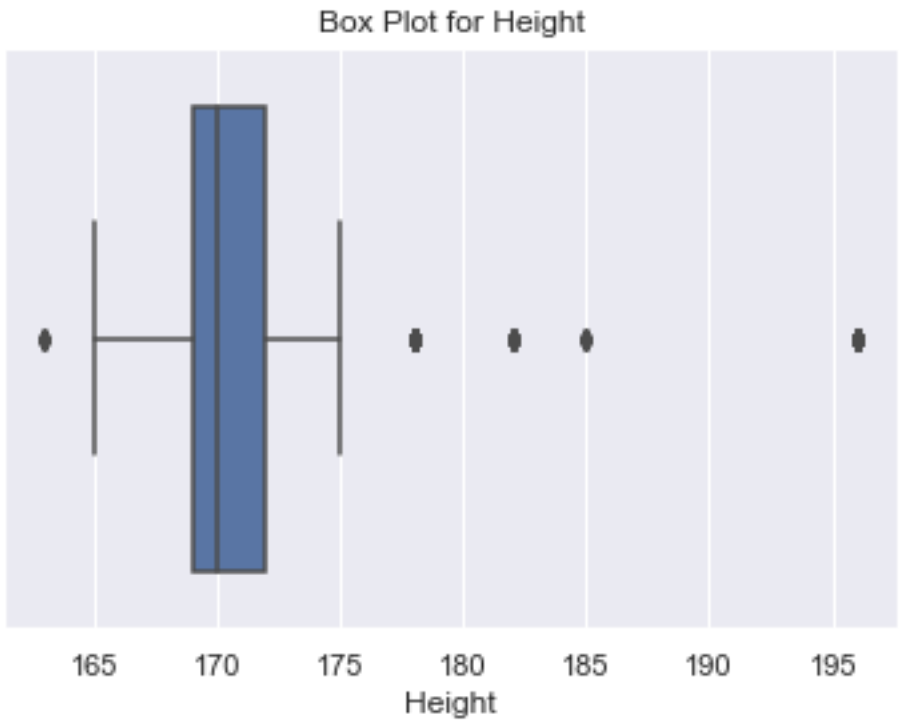

Box Plot for Service time
0
5
10
15
20
25
30
Service time

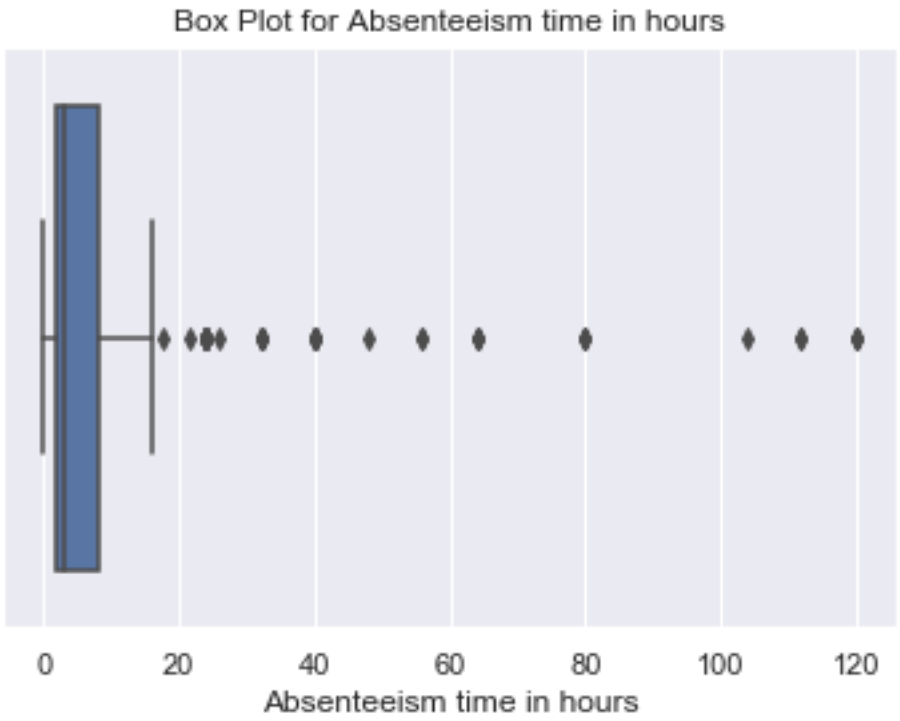

Box Plot for Age
30
35
40
45
50
55
Age


Box Plot for Work load Average/day
200000
225000
250000
275000
300000
325000
350000
375000
Work load Average/day


Box Plot for Hit target
82.5
85.0
87.5
90.0
92.5
95.0
97.5
100.0
Hit target


Box Plot for Pet
0
1
2
3
4
5
6
7
8
Pet


Box Plot for Height
165
170
175
180
185
190
195
Height



The above boxplots clearly shows the outliers present in the variables. So to treat the outlier we calculated upper and lower quartile then the interquartile range and then we created nan value in place of outliers which are later replaced by knn imputation method.

Boxplots after outlier treatment

```
#detect and replace outliers with NA
#Extracting quartiles
for i in variable_num:
    q75,q25=np.percentile(emp[i],[75,25])



   ##calculating iqr
    iqr=q75-q25

    #calculating inner and outer fence
    minimum= q25-(iqr*1.5)
    maximum= q75+(iqr*1.5)

    #replace with NA
    emp.loc[emp[i]<minimum,i] = np.nan
    emp.loc[emp[i]>maximum,i] = np.nan
```

```
emp.isnull().sum()
```

```
ID                               0
Reason for absence               0
Month of absence                 0
Day of the week                  0
Seasons                          0
Transportation expense           3
Distance from Residence to Work  0
Service time                     5
Age                              8
Work load Average/day           31
Hit target                      19
Disciplinary failure             0
Education                        0
Son                              0
Social drinker                   0
Social smoker                    0
Pet                             46
Weight                           0
Height                         119
Body mass index                  0
Absenteeism time in hours       46
dtype: int64
```

```
]: #Impute with KNN
emp=pd.DataFrame(KNN(k=3).fit_transform(emp), columns = emp.columns)
```

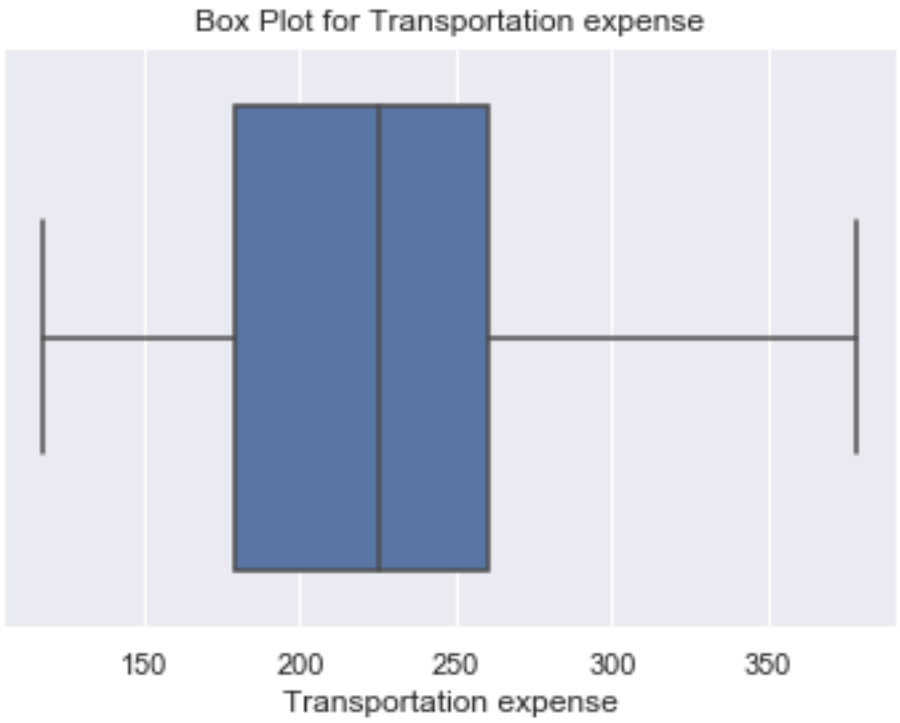
```
Imputing row 1/740 with 0 missing, elapsed time: 0.111
Imputing row 101/740 with 1 missing, elapsed time: 0.111
Imputing row 201/740 with 1 missing, elapsed time: 0.113
Imputing row 301/740 with 0 missing, elapsed time: 0.113
Imputing row 401/740 with 0 missing, elapsed time: 0.113
Imputing row 501/740 with 0 missing, elapsed time: 0.117
Imputing row 601/740 with 0 missing, elapsed time: 0.117
Imputing row 701/740 with 0 missing, elapsed time: 0.117
```

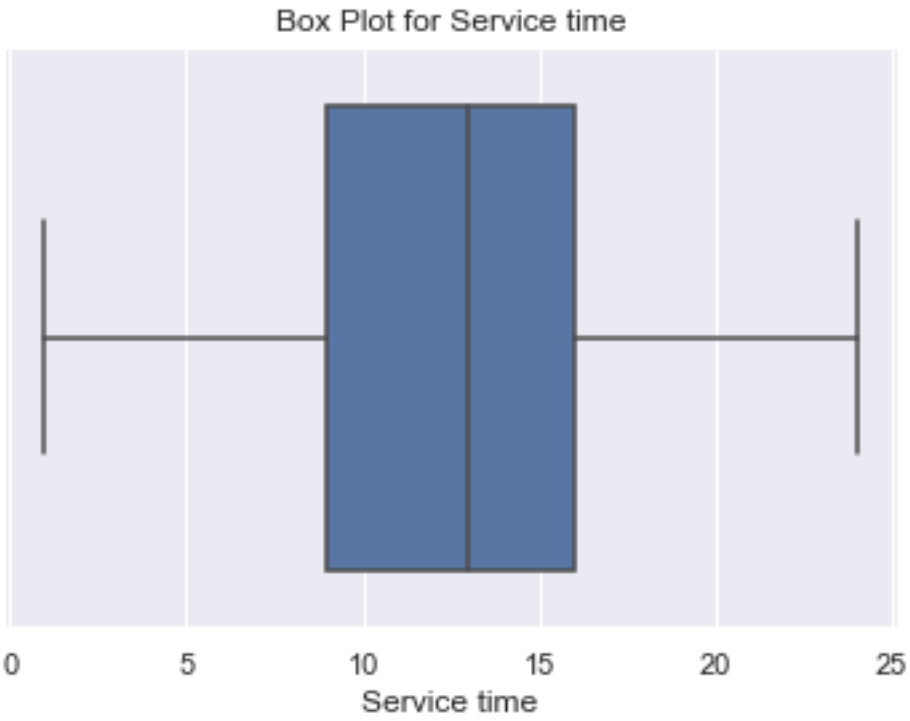
```
]: emp.isnull().sum()
```

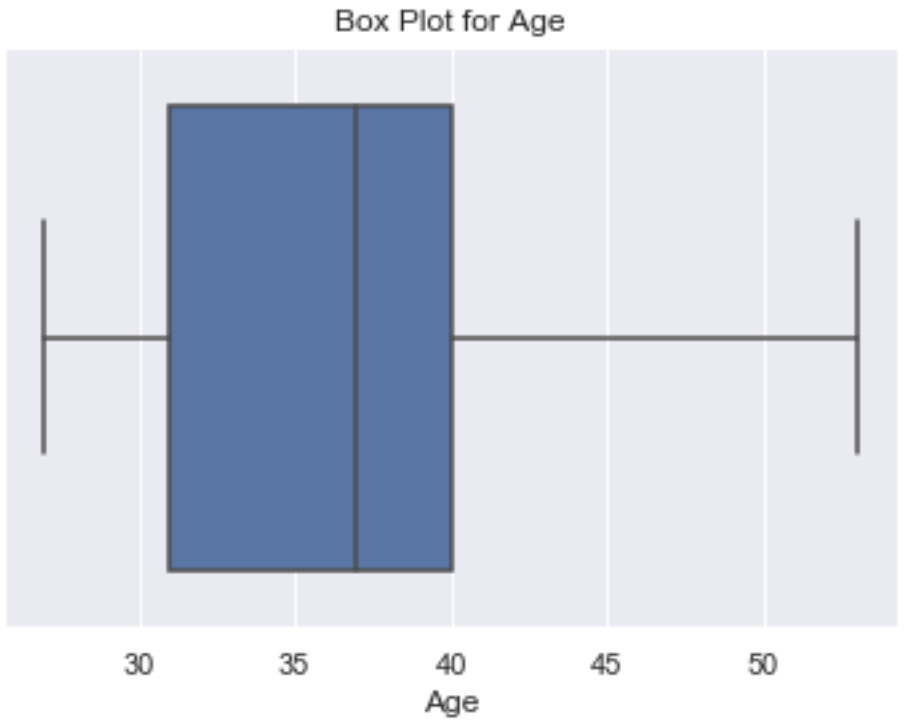
```
emp.isnull().sum()
```

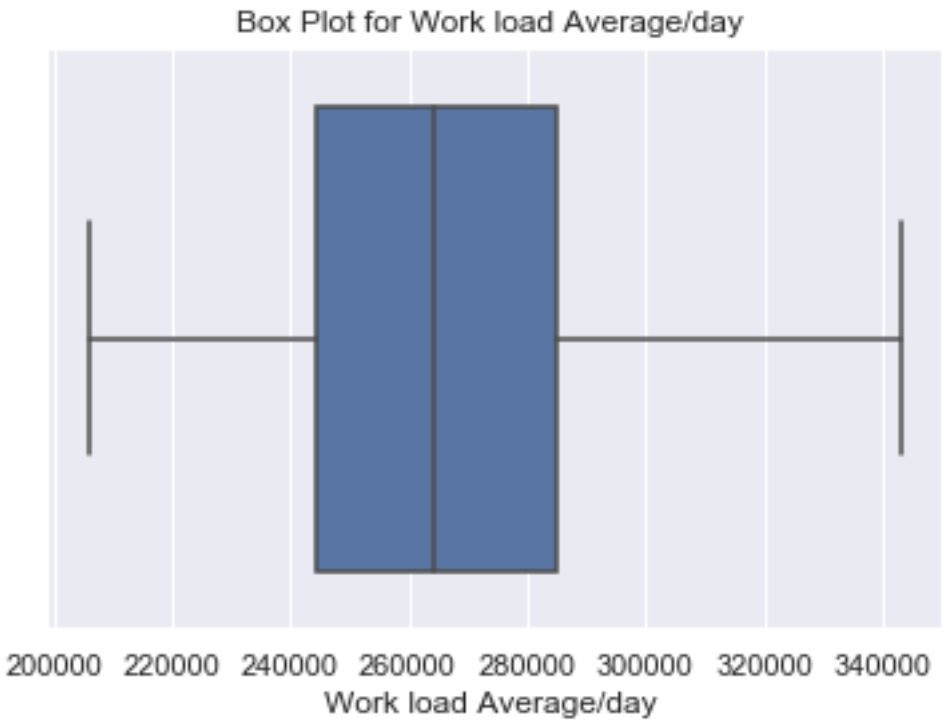
```
ID                               0
Reason for absence               0
Month of absence                 0
Day of the week                  0
Seasons                          0
Transportation expense           0
Distance from Residence to Work  0
Service time                     0
Age                              0
Work load Average/day            0
Hit target                       0
Disciplinary failure             0
Education                        0
Son                              0
Social drinker                   0
Social smoker                    0
Pet                              0
Weight                           0
Height                           0
Body mass index                  0
Absenteeism time in hours        0
dtype: int64
```

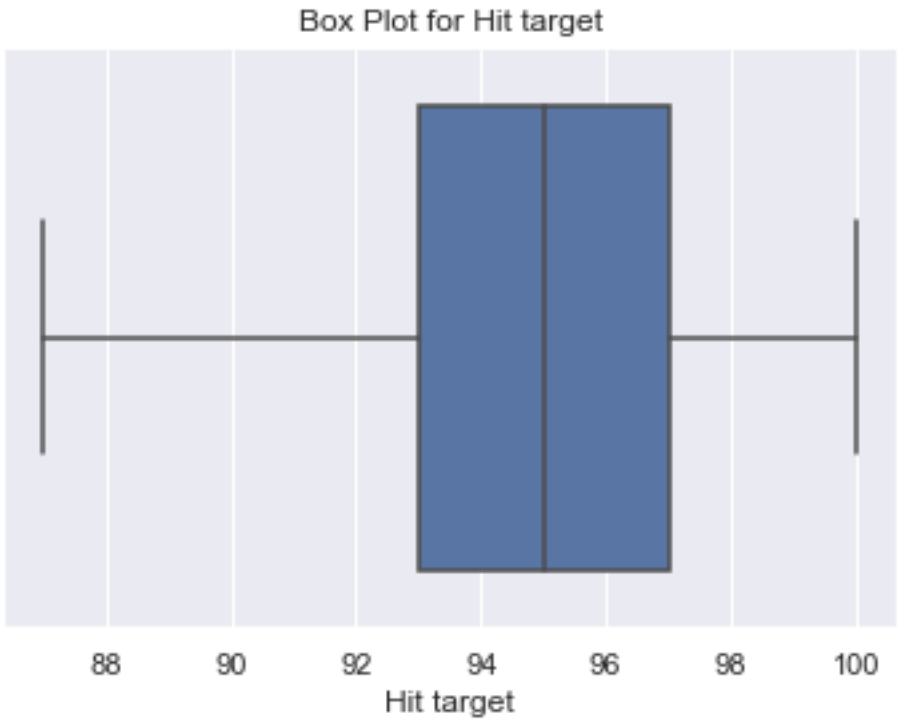
```
#Draw box plot
for i in variable_num:
    sns.boxplot(emp[i])
    plt.title("Box Plot for "+str(i))
    plt.show()
```

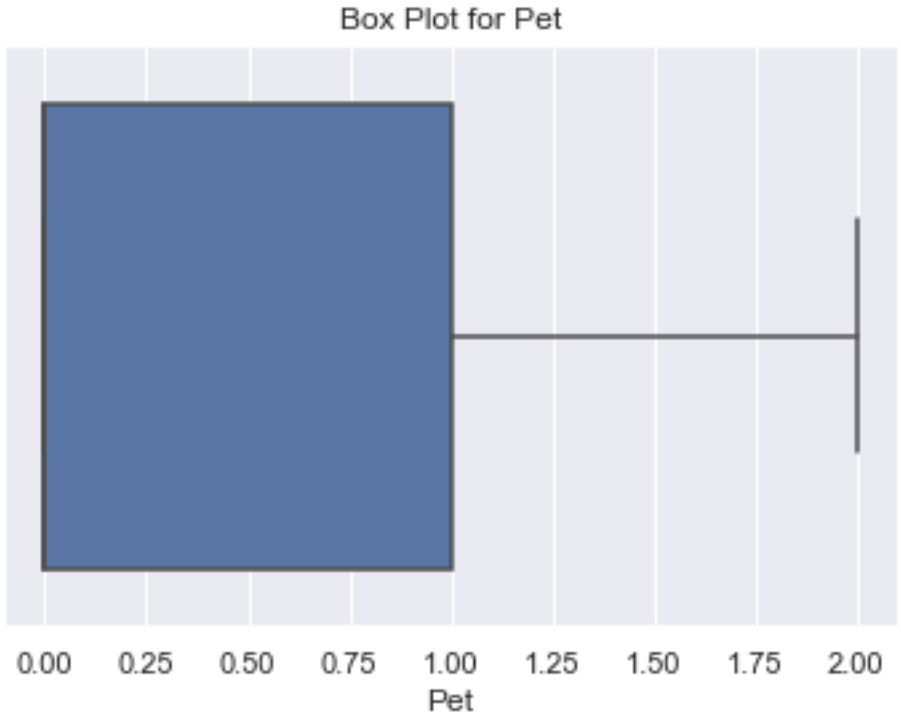


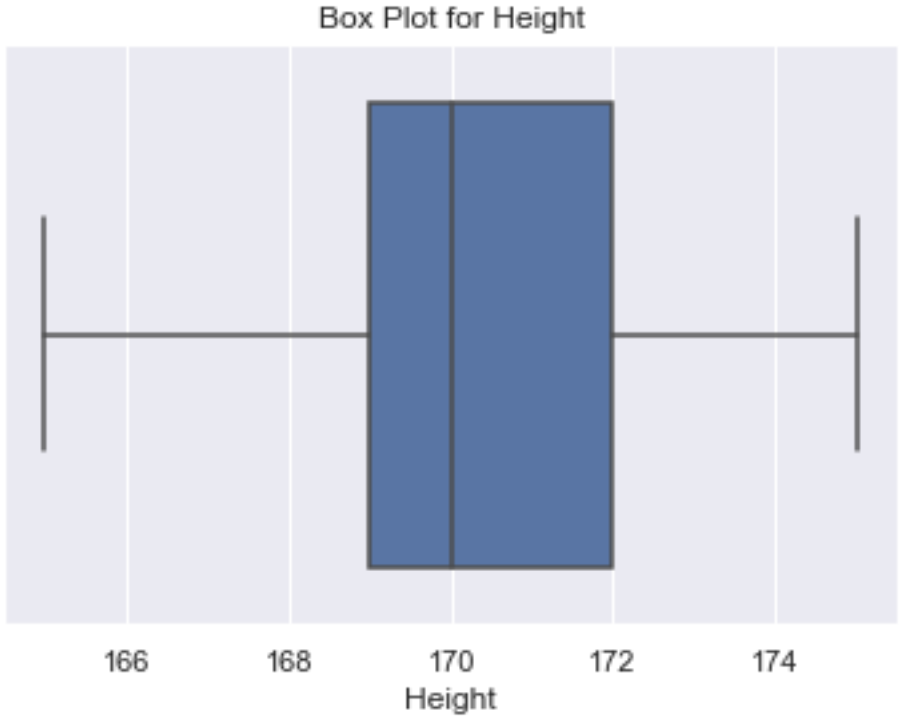

Box Plot for Service time
0
5
10
15
20
25
Service time


Box Plot for Age
30
35
40
45
50
Age


Box Plot for Work load Average/day
200000
220000
240000
260000
280000
300000
320000
340000
Work load Average/day


Box Plot for Hit target
88
90
92
94
96
98
100
Hit target





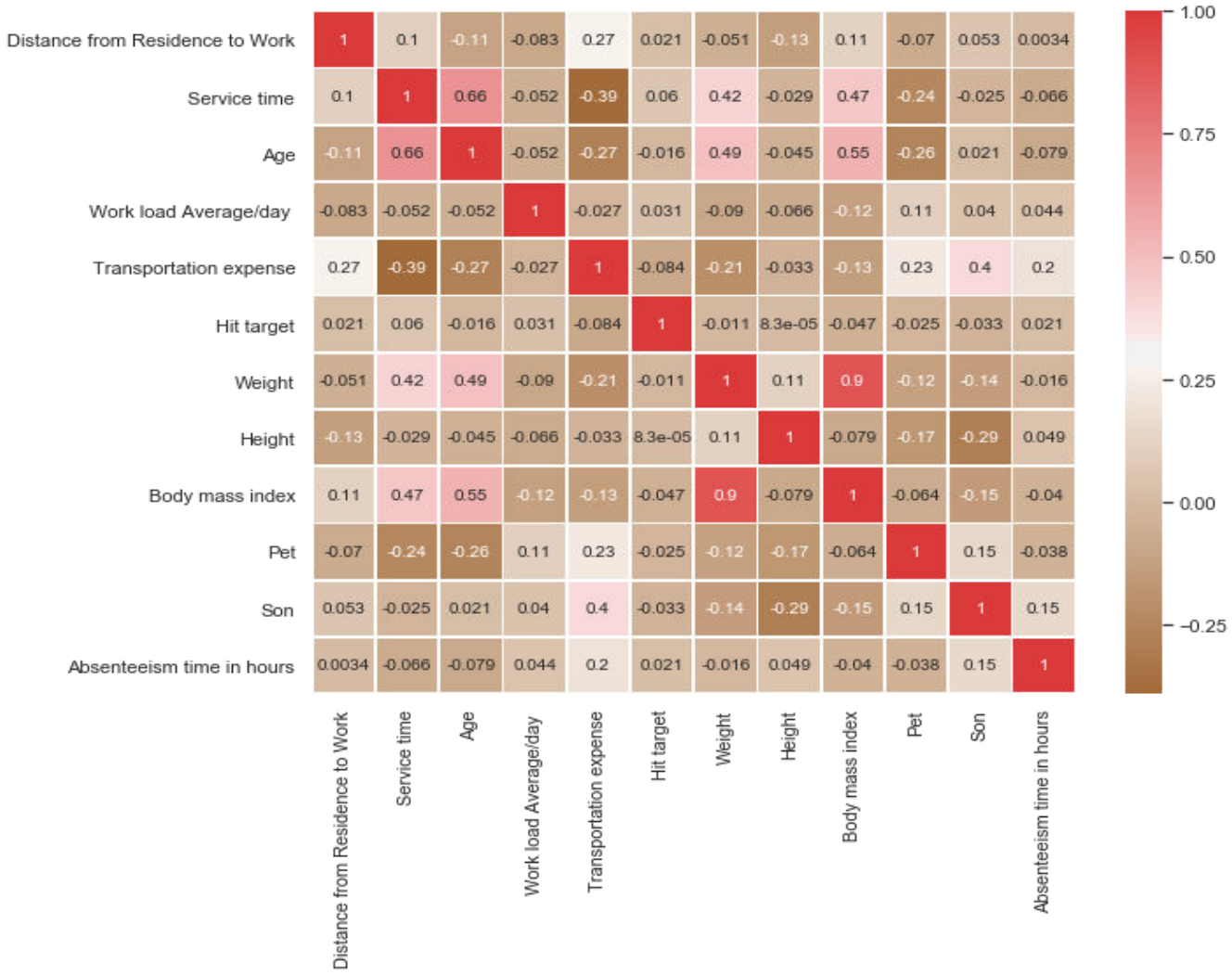
2.1.3 Feature Selection

Machine learning works on a simple rule of GIGO i.e. Garbage In Garbage Out. Here garbage refers to the noise or redundant values.

This becomes even more important when the number of features are very large. We need not use every feature at our disposal for creating an algorithm. We can assist our algorithm by feeding in only those features that are important. Feature subsets gives better results than complete set of features for the same algorithm or “Sometimes, less is better!”.

We should consider the selection of feature for model keeping in mind that there should be low correlation between two independent variables otherwise there will be problem of multicollinearity.

Fig. 3.0



From above correlation plot we can clearly see that variable Weight and Body mass index have high correlation variable Age and Service Time also have high correlation. It means that we must drop one variable out of two having high correlation. So in our study here we will drop variables Weight and Age.

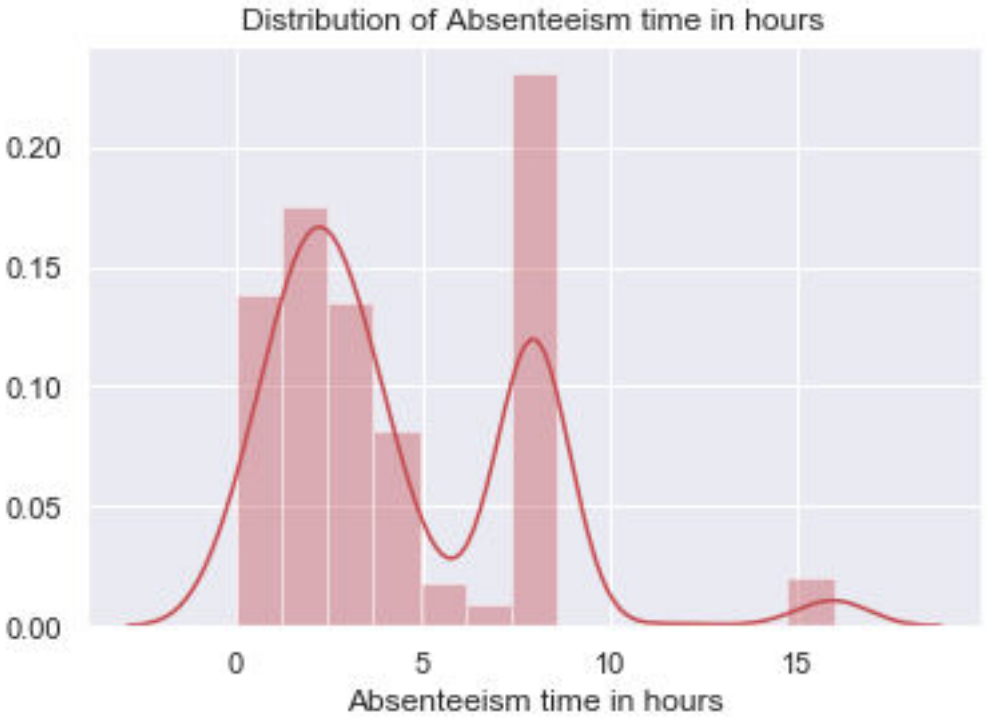
Color dark Red indicates there is strong positive correlation and if dark brown indicates negative correlation.

3. Data Distribution

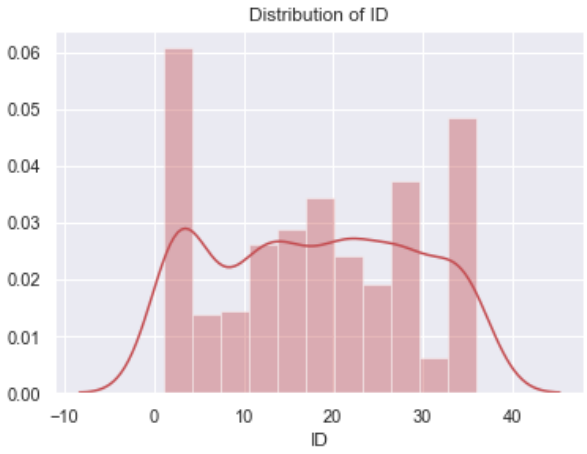
Checking distribution of variables with help of distribution plot

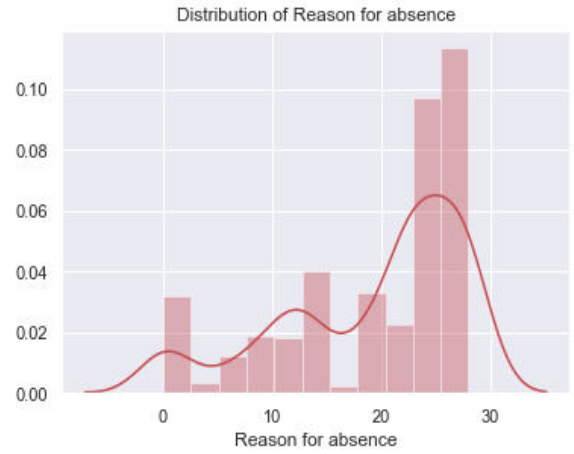
Distribution of target variable (Absenteeism time in hours)

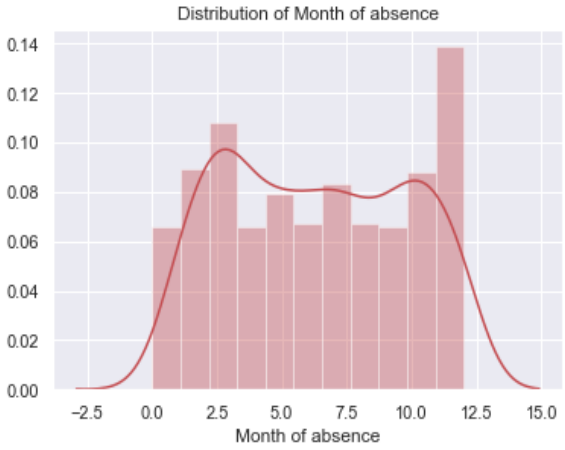
Fig. 4.0

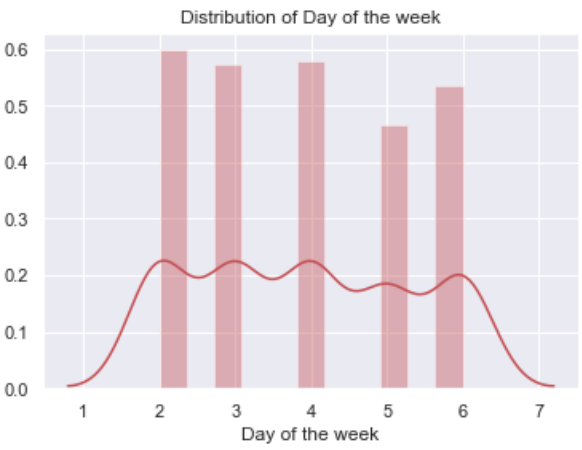


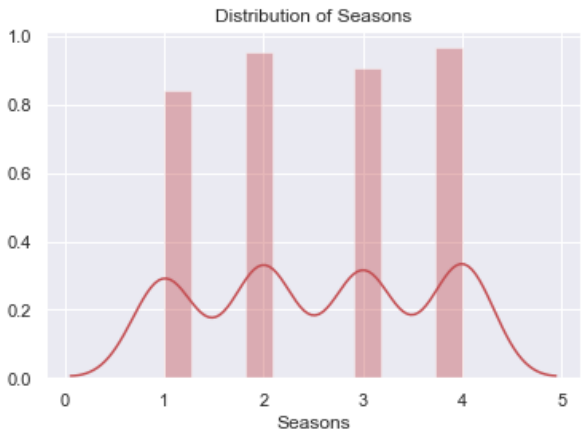
From above grapgh we can see that our target variable Absenteeism time in hours is not normally distributed.

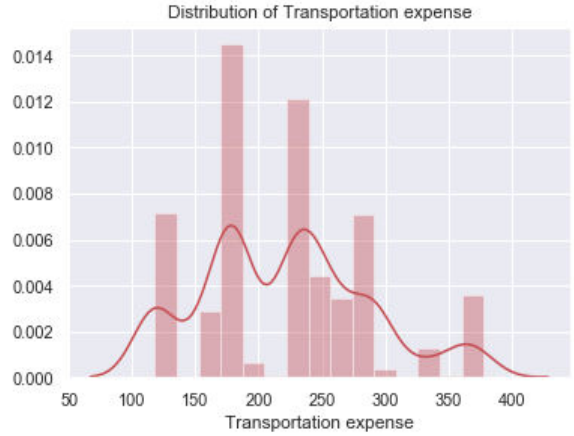

Distribution of ID
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−10
0
10
20
30
40
ID

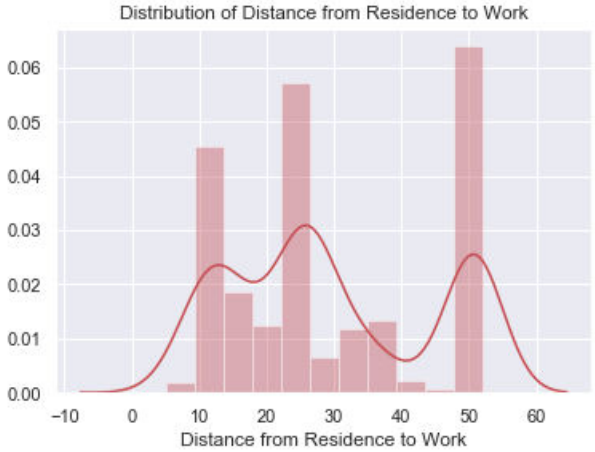

Distribution of Reason for absence
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
Reason for absence

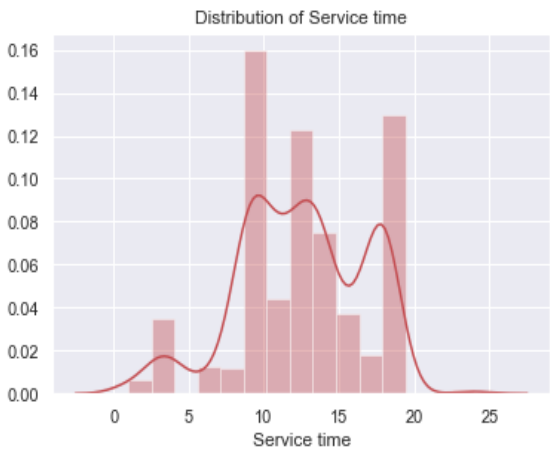

Distribution of Month of absence
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
Month of absence

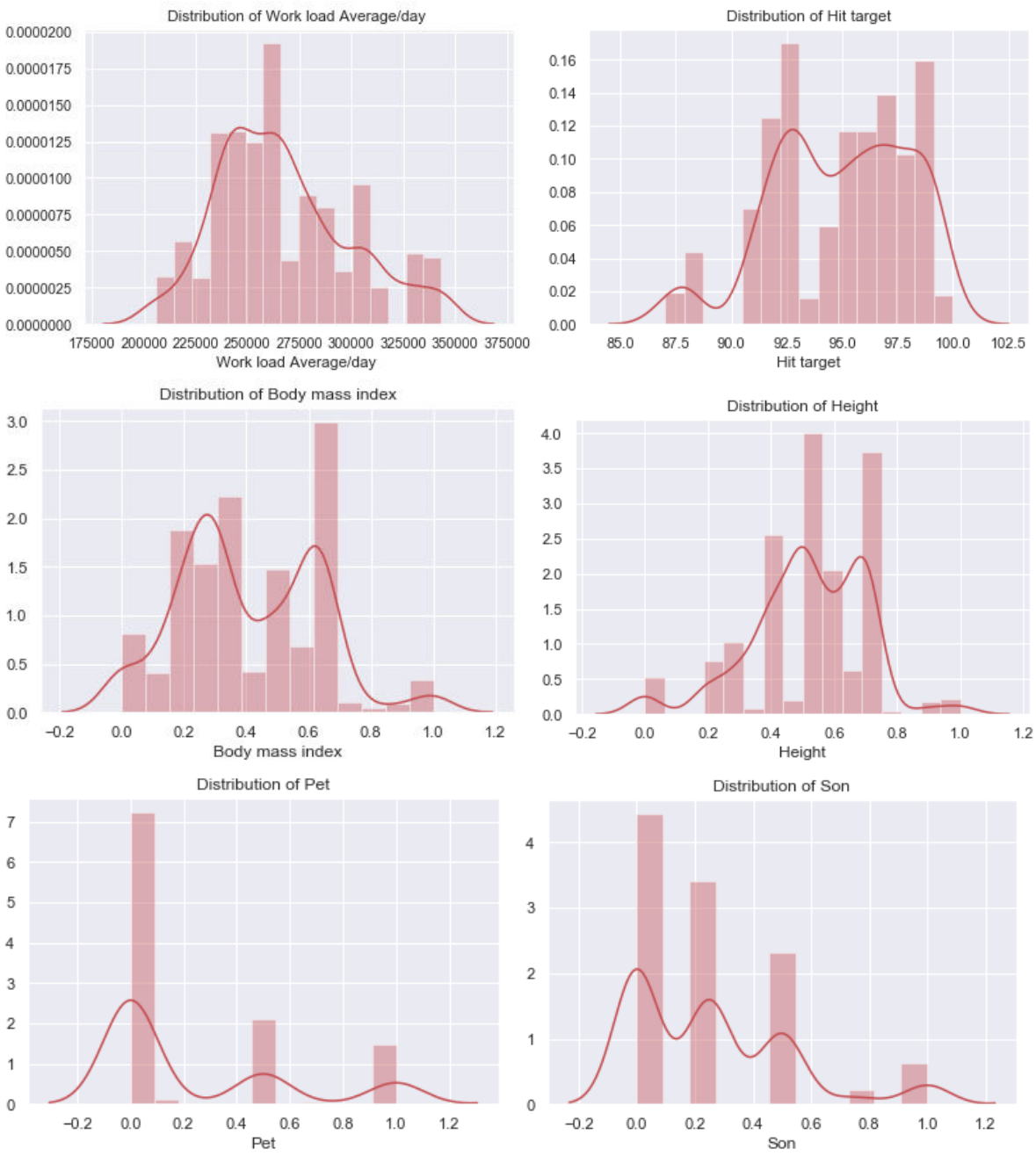

Distribution of Day of the week
0.6
0.5
0.4
0.3
0.2
0.1
0.0
1
2
3
4
5
6
7
Day of the week


Distribution of Seasons
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
Seasons


Distribution of Transportation expense
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
50
100
150
200
250
300
350
400
Transportation expense


Distribution of Distance from Residence to Work
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−10
0
10
20
30
40
50
60
Distance from Residence to Work


Distribution of Service time
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20
25
Service time



From above distribution plots we can clearly see that data is not normaly distributed. So, we will normalize data i.e. we will bring in range of 0 to 1 So that data gets normally distributed.

```
#Since we have dropped few variables and stored in emp_reduced. So here we have to updated the variable
#and store them in new object.
variable_num_update = ['Distance from Residence to Work','Pet','Son',
                       'Work load Average/day ', 'Transportation expense',
                       'Hit target', 'Height','Service time',
                       'Body mass index', 'Absenteeism time in hours']

variable_cat_update = ['ID','Reason for absence','Disciplinary failure','Social drinker','Day of the week','Education',
                       'Month of absence','Social smoker','Seasons']
```

```
#Since data is not normally distributed hence we will normalize our data for further analysis
for i in variable_num_update:
    print(i)
    emp_reduced[i] = (emp_reduced[i] - emp_reduced[i].min())/(emp_reduced[i].max()-emp_reduced[i].min())
```

```
Distance from Residence to Work
Pet
Son
Work load Average/day
Transportation expense
Hit target
Height
Service time
Body mass index
Absenteeism time in hours
```

4. Dummies Creation

Since we have categorical variable also in our data set and we can't make calculation on data having categorical variable. So, we have made use of dummy variable for categories which helps us to make calculations and build suitable model.

Creating Dummies

```
#Save target variable first
emp_dummy = pd.DataFrame(emp_reduced['Absenteeism time in hours'])
```

```
emp_dummy.shape
```

```
(740, 1)
```

```
#here in variable_num_update1 we have not added 'Absenteeism time in hours' as we have already added in data frame and adding it
#again will lead to overlap error.
variable_num_update1 = ['Distance from Residence to Work','Pet','Son',
                        'Work load Average/day ', 'Transportation expense',
                        'Hit target', 'Height','Service time',
                        'Body mass index']
```

```
variable_cat_update1 = ['ID','Reason for absence','Disciplinary failure','Social drinker','Day of the week','Education',
                        'Social smoker','Month of absence','Seasons',]
```

```
emp_dummy.head(2)
```

	Absenteeism time in hours
0	0.25
1	0.00

```
##Create dummies for categorical variables. Above we have already stored categorical variable in variable_cat

for i in variable_cat_update1:
    temp = pd.get_dummies(emp_reduced[i], prefix = i)
    emp_dummy = emp_dummy.join(temp)
```

```
emp_dummy.head(5)
```

	Absenteeism time in hours	Distance from Residence to Work	Pet	Son	Work load Average/day	Transportation expense	Hit target	Height	Service time	Body mass index	...	Month of absence_8.0	Month of absence_9.0	Month of absence_10.0	abse
0	0.250	0.659574	0.5	0.50	0.244925	0.657692	0.769231	0.700000	0.521739	0.578947	...	0	0	0	
1	0.000	0.170213	0.0	0.25	0.244925	0.000000	0.769231	0.500001	0.739130	0.631579	...	0	0	0	
2	0.125	0.978723	0.0	0.00	0.244925	0.234615	0.769231	0.500000	0.739130	0.631579	...	0	0	0	
3	0.250	0.000000	0.0	0.50	0.244925	0.619231	0.769231	0.300000	0.565217	0.263158	...	0	0	0	
4	0.125	0.659574	0.5	0.50	0.244925	0.657692	0.769231	0.700000	0.521739	0.578947	...	0	0	0	

5 rows × 118 columns

```
emp_dummy.shape
```

```
(740, 118)
```

5. Modelling

5.1 Model Selection

Model Selection is a process of selecting the model which have better accuracy and can work on train and test data. We must select a model where algorithm works well and shows low error rate. We made Decision Tree, Random Forest and Linear regression model. Previously when we ran all three models the results were not so good. So we made use Principal component analysis which helped us to select the variables that explained variance of data. It happens that at a certain point the variables don't show much of variance in data and can lead to complexity so in such case the PCA helps to select variables that actually show the variance.

5.1.1 Principal Component Analysis (PCA)

Dimensionality reduction using PCA

```
]: # Dimensionality reduction by using pca
   emp1 = emp_dummy['Absenteeism time in hours']
```

```
]: #emp_dummy.drop(['Absenteeism time in hours'], inplace = True, axis=1)
   emp_dummy.shape
```

```
]: (740, 118)
```

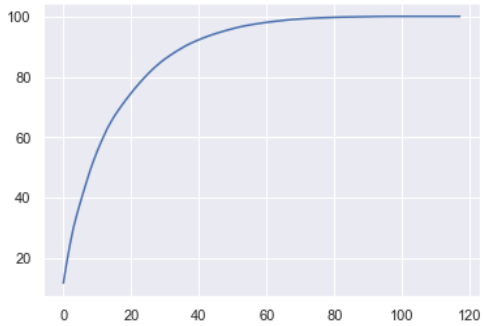
```
]: from sklearn.decomposition import PCA

   # Converting data to numpy array
   X = emp_dummy.values
```

```
]: # Data has 118 variables so no of components of PCA = 118
   pca = PCA(n_components=118)
   pca.fit(X)
```

```
]: PCA(copy=True, iterated_power='auto', n_components=118, random_state=None,
     svd_solver='auto', tol=0.0, whiten=False)
```

```
]: # The amount of variance that each PC explains
   var= pca.explained_variance_ratio_
```

```
# Cumulative Variance explains
sns.set()
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
plt.plot(var1)
plt.show()
```




```
# From the above plot selecting 50 components since it explains almost 95+ % data variance
pca = PCA(n_components=50)


# Fitting the selected components to the data
pca.fit(X)
```

5.1.2 Decision Tree

A tree has many analogies in real life and turns out that it has influenced a wide area of **machine learning**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

Dividing data into train ¶

```
# Using train_test_split sampling function for test and train data split
X_train, X_test, y_train, y_test = train_test_split(X,emp1, test_size=0.2)
```

Decision Tree

```
# Building model on top of training dataset
fit_DT = DecisionTreeRegressor(max_depth = 5).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)))
```

```
Root Mean Squared Error For Training data = 0.0014547258030807727
Root Mean Squared Error For Test data = 0.009282941496556348
R^2 Score(coefficient of determination) = 0.9984520572905576
```

Root Mean Squared Error For Training data = 0.0014547258030807727
Root Mean Squared Error For Test data = 0.009282941496556348
R^2 Score(coefficient of determination) = 0.9984520572905576

5.1.3 Random Forest

Random forests is an ensemble learning method for classification and regression that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Random forest functions in following way

- Draws a bootstrap sample from training data.
- For each sample grow a decision tree and at each node of the tree

a. Ramdomly draws a subset of mtry variable and p total of features that are available
b. Picks the best variable and best split from the subset of mtry variable
c. Continues until the tree is fully grown.

Random Forest Implementation

Random Forest

```
5]: #Random Forest
    RF_model = RandomForestRegressor(n_estimators = 500, oob_score = True, n_jobs = -1,random_state =50,max_features = "auto",
                                     min_samples_leaf = 50).fit(X_train, y_train)
```

```
7]: # Calculating RMSE for training data to check for over fitting
    pred_train = RF_model.predict(X_train)
    rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
```

```
3]: # Calculating RMSE for test data to check accuracy
    pred_test = RF_model.predict(X_test)
    rmse_for_test =np.sqrt(mean_squared_error(y_test,pred_test))
```

```
9]: 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)))
```

```
Root Mean Squared Error For Training data = 0.08463919371751547
Root Mean Squared Error For Test data = 0.07135560349067667
R^2 Score(coefficient of determination) = 0.8598614268421698
```

Root Mean Squared Error For Training data = 0.08463919371751547
Root Mean Squared Error For Test data = 0.07135560349067667
R^2 Score(coefficient of determination) = 0.8598614268421698

5.1.4 Linear Regression

Multiple linear regression is the most common form of linear regression analysis. As a predictive analysis, the multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables. The independent variables can be continuous or categorical.

Linear Regression

```
]: # 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)))
```

```
Root Mean Squared Error For Training data = 4.529890363154557e-16
Root Mean Squared Error For Test data = 0.00030505994938045586
R^2 Score(coefficient of determination) = 0.9999979137208953
```

```
Root Mean Squared Error For Training data = 4.529890363154557e-16
Root Mean Squared Error For Test data = 0.00030505994938045586
R^2 Score(coefficient of determination) = 0.9999979137208953
```

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. 0% indicates that the model explains none of the variability of the response data around its mean.

As R-Square value is 0.9999 and root mean squared error is also low both for training and test data hence linear regression model is well suited.

6. Conclusion: - For the Employee Absenteeism Linear regression Model is best model to predict the count.

Appendix - A

Answer to first question monthly absenteeism in hours

Finding Average monthly hours of Absenteeism

```
hr_loss = pd.DataFrame(emp[['Month of absence','Absenteeism time in hours']])
```

```
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(1, "January")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(2, "Ferbruary")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(3, "March")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(4, "April")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(5, "May")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(6, "june")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(7, "July")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(8, "August")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(9, "September")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(10, "October")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(11, "November")
hr_loss['Month of absence'] = hr_loss['Month of absence'].replace(12, "December")
```

```
hr_loss.head(5)
```

	Month of absence	Absenteeism time in hours
0	July	4.0
1	July	0.0
2	July	2.0
3	July	4.0
4	July	2.0

```
hr_loss.groupby('Month of absence')['Absenteeism time in hours'].mean()
```

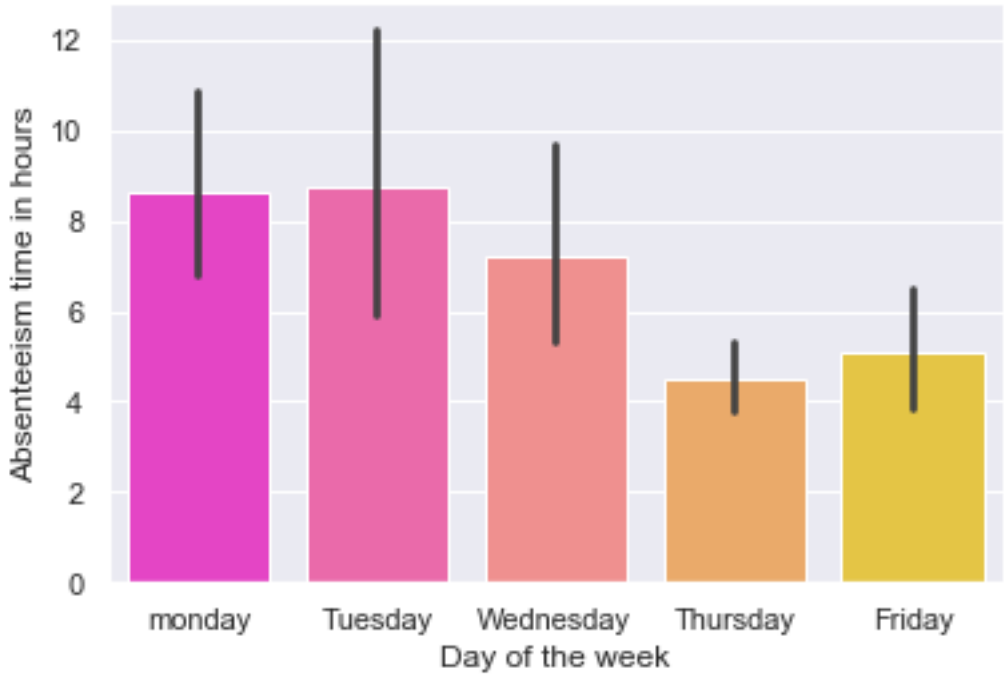
```
Month of absence
0.0                 0.000000
10.00000040690432   3.000000
April               4.484123
August              4.649133
December            3.999068
Ferbruary           3.849183
January             3.491295
July                5.613350
March               5.259064
May                 4.162869
November            4.134807
October             4.242845
September           3.836394
june                4.551624
Name: Absenteeism time in hours, dtype: float64
```

```
hr_loss.isnull().sum()
```

```
Month of absence            0
Absenteeism time in hours   0
dtype: int64
```

Answer to second question where company must take action

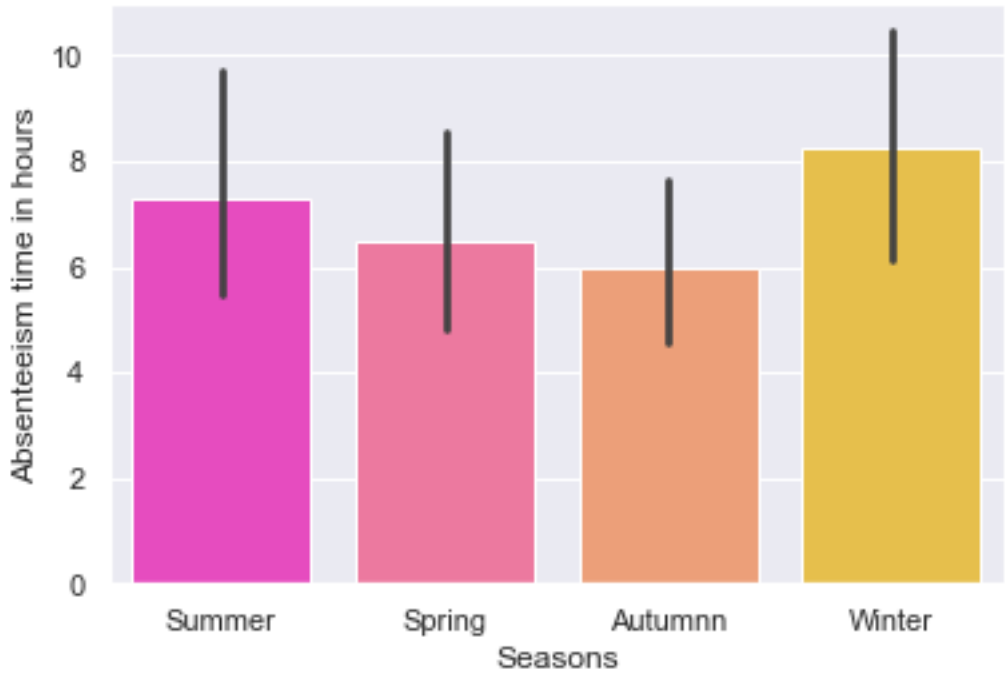
Bar plot between Day of week and Absenteeism in hours




```
sns.barplot(x='Day of the week', y='Absenteeism time in hours', palette = 'spring', data = emp, order  =['monday','Tuesday',
                                                                                                          'Wednesday','Thursday','
```

From above bar plot we can see that on Tuesdays the absenteeism hours are highest. The company should see that why on Tuesday's absenteeism time is so high.

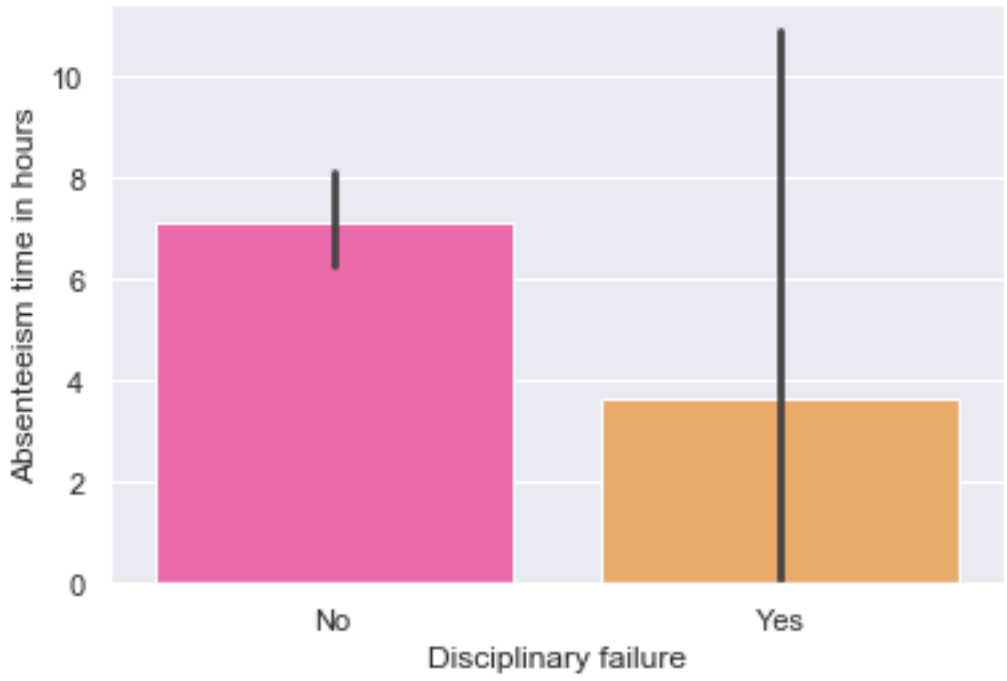
Bar plot between Seasons and Absenteeism time in hours




```
sns.barplot(x='Seasons', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

As we can see from above bar plot Absenteeism time in hours is more in winter season.

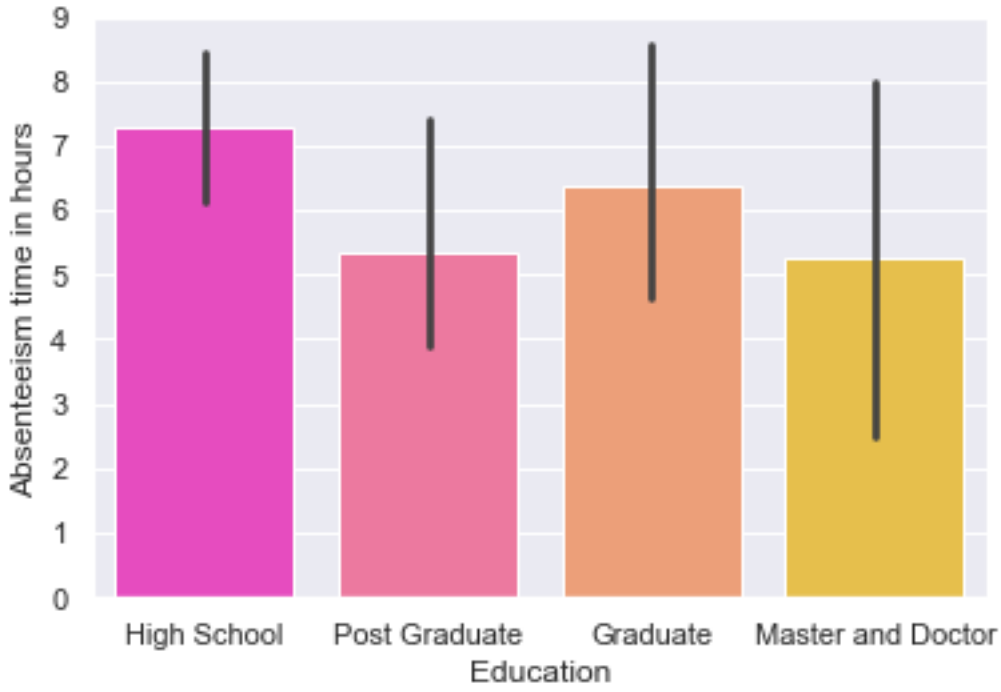
Bar plot between Disciplinary failure and Absenteeism time in hours




```
sns.barplot(x='Disciplinary failure', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

From above bar plot we can see that Disciplinary failure is not the reason for Absenteeism in hours.

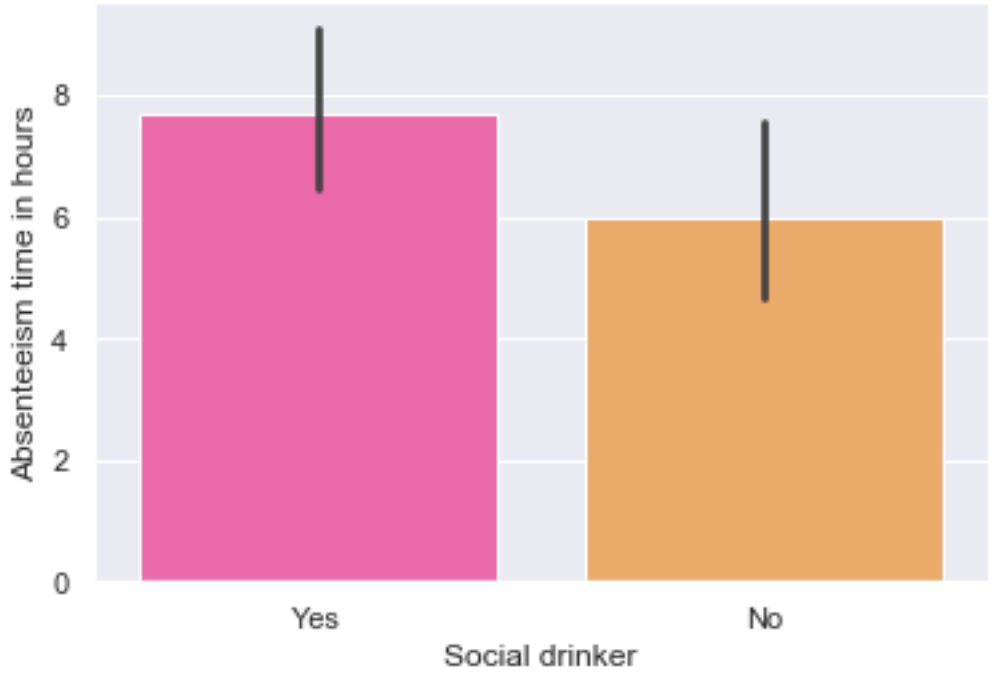
Bar plot between Education and Absenteeism time in hours




```
sns.barplot(x='Education', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

We can see from above bar plot that those who have education of high school have highest absenteeism time in hours which means company should watch them and should take action against them.

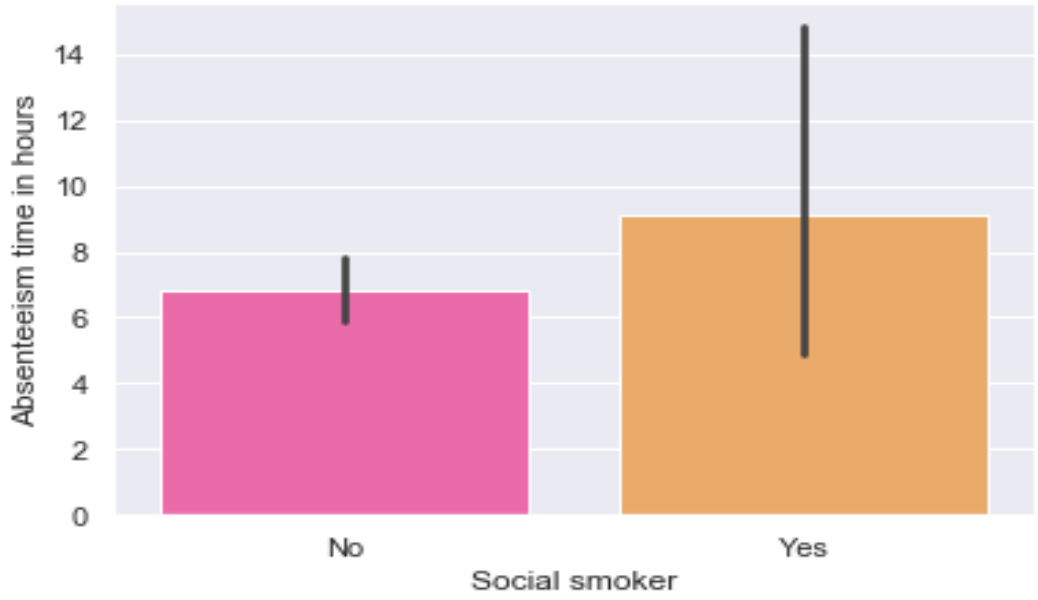
Bar plot between Social drinker and Absenteeism time in hours




```
sns.barplot(x='Social drinker', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

From above plot we can clearly see that those who are social drinker are having a greater number of absenteeism hours which means the company should warn them for drinking habit that should not affect the company's work and performance.

Bar plot between Social smoker and Absenteeism time in hours




```
sns.barplot(x='Social smoker', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

From above plot we can clearly see that those who are social smoker are having a greater number of absenteeism hours which means the company should warn them for smoking habit that should not affect the company's work and performance.

Appendix- B - Python Code

Fig 3.0 Python Code

Feature Selection

```
br_corr = emp.loc[:,variable_num]
```

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

corr_matrix = br_corr.corr()

sns.heatmap(corr_matrix, mask = np.zeros_like(corr_matrix, dtype=np.bool), cmap = sns.diverging_palette(400,12, as_cmap= True),
            annot = True , linewidths = 0.9,square = True, ax=ax)
```

Complete Python File

Loading libraries

```
#Loading libraries
import os
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
from ggplot import *
import matplotlib.pyplot as plt
from fancyimpute import KNN
from scipy import stats
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
```

```
#Setting working directory
os.chdir('E:/Project/Employee absentism/Python')
```

```
os.getcwd()
```

```
'E:\\Project\\Employee absentism\\Python'
```

```
#Loading dataset file
emp = pd.read_csv('E:/Project/Employee absentism/Python/employee_absenteeism.csv')
```

```
df=emp.copy()
```

```
emp.head(5)
```

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	...	Disciplinary failure	Education	Son	Social drinker	Social smoker	Pet	Wei
0	11	26.0	7.0	3	1	289.0	36.0	13.0	33.0	239554.0	...	0.0	1.0	2.0	1.0	0.0	1.0	9
1	36	0.0	7.0	3	1	118.0	13.0	18.0	50.0	239554.0	...	1.0	1.0	1.0	1.0	0.0	0.0	9
2	3	23.0	7.0	4	1	179.0	51.0	18.0	38.0	239554.0	...	0.0	1.0	0.0	1.0	0.0	0.0	8
3	7	7.0	7.0	5	1	279.0	5.0	14.0	39.0	239554.0	...	0.0	1.0	2.0	1.0	1.0	0.0	6
4	11	23.0	7.0	5	1	289.0	36.0	13.0	33.0	239554.0	...	0.0	1.0	2.0	1.0	0.0	1.0	9

5 rows × 21 columns

```
#checking the data types
emp.dtypes
```


```
#checking the data types
emp.dtypes
```


```
ID                               int64
Reason for absence             float64
Month of absence               float64
Day of the week                  int64
Seasons                          int64
Transportation expense         float64
Distance from Residence to Work    float64
Service time                   float64
Age                            float64
Work load Average/day          float64
Hit target                     float64
Disciplinary failure           float64
Education                      float64
Son                            float64
Social drinker                 float64
Social smoker                  float64
Pet                            float64
Weight                         float64
Height                         float64
Body mass index                float64
Absenteeism time in hours      float64
dtype: object
```

Missing Value analysis

```
emp.isnull().sum()
```

```
ID                               0
Reason for absence               3
Month of absence                 1
Day of the week                  0
Seasons                          0
Transportation expense           7
Distance from Residence to Work    3
Service time                     3
Age                              3
Work load Average/day           10
Hit target                       6
Disciplinary failure             6
Education                       10
Son                              6
Social drinker                   3
Social smoker                    4
Pet                              2
Weight                           1
Height                          14
Body mass index                 31
Absenteeism time in hours       22
dtype: int64
```

```
#From above value we can clearly see that there are missing values. since there are missing values present we need to impute
#these missing values.
```

```
emp.shape
```

```
(740, 21)
```

```
#create data frame with missing values
emp_missing_value = pd.DataFrame(emp.isnull().sum())
```

```
emp_missing_value
```

	0
ID	0
Reason for absence	3
Month of absence	1
Day of the week	0
Seasons	0
Transportation expense	7
Distance from Residence to Work	3
Service time	3
Age	3
Work load Average/day	10
Hit target	6
Disciplinary failure	6
Education	10
Son	6
Social drinker	3
Social smoker	4
Pet	2
Weight	1
Height	14
Body mass index	31
Absenteeism time in hours	22

```
#Reset index
emp_missing_value = emp_missing_value.reset_index()
```

```
emp_missing_value
```

	index	0
0	ID	0
1	Reason for absence	3
2	Month of absence	1
3	Day of the week	0
4	Seasons	0
5	Transportation expense	7
6	Distance from Residence to Work	3
7	Service time	3
8	Age	3
9	Work load Average/day	10
10	Hit target	6
11	Disciplinary failure	6
12	Education	10
13	Son	6
14	Social drinker	3
15	Social smoker	4
16	Pet	2
17	Weight	1
18	Height	14
19	Body mass index	31
20	Absenteeism time in hours	22

```
#Renaming the columns
emp_missing_value = emp_missing_value.rename(columns = {'index':'variables',0:'missing_percentage'})
```

```
emp_missing_value
```

```
#calculating percentage
emp_missing_value['missing_percentage'] = (emp_missing_value['missing_percentage']/len(emp))*100
```

```
emp_missing_value
```

	variables	missing_percentage
0	ID	0.000000
1	Reason for absence	0.405405
2	Month of absence	0.135135
3	Day of the week	0.000000
4	Seasons	0.000000
5	Transportation expense	0.945946
6	Distance from Residence to Work	0.405405
7	Service time	0.405405
8	Age	0.405405
9	Work load Average/day	1.351351
10	Hit target	0.810811
11	Disciplinary failure	0.810811
12	Education	1.351351
13	Son	0.810811
14	Social drinker	0.405405

```
emp_missing_value.to_csv('E:/Project/Employee absentism/Python/missing_value_perc.csv')
```

```
#Creating missing value in Transportation expense and later imputing it.
```

```
#Creating copy
df = emp.copy()
```

```
df1=emp.copy()
```

```
#Creating missing value
emp['Transportation expense'].iloc[10]
```

```
260.0
```

```
#Imputation method
#actual value = 260
#Mean = 220.98
#Median = 225
#KNN = 259.99
```

```
emp['Transportation expense'].iloc[10] = np.nan
```

```
emp['Transportation expense'].iloc[10]
```

```
nan
```

```
#Impute with mean
emp['Transportation expense'] = emp['Transportation expense'].fillna(emp['Transportation expense'].mean())
```

```
emp['Transportation expense'].iloc[10]
```

```
220.98224043715848
```

```
#Loading data set again
emp = df.copy()
```

```
emp['Transportation expense'].iloc[10]
```

```
260.0
```

```
emp['Transportation expense'].iloc[10] = np.nan
```

```
C:\Users\Aditya\Anaconda3\lib\site-packages\pandas\core\indexing.py:189: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self._setitem_with_indexer(indexer, value)
```

```
emp['Transportation expense'].iloc[10]
```

```
nan
```

```
#Impute with median
emp['Transportation expense'] = emp['Transportation expense'].fillna(emp['Transportation expense'].median())
```

```
emp['Transportation expense'].iloc[10]
```

```
225.0
```

```
##impute with KNN
```

```
#Loading dataaset again
emp= df1.copy()
```

```
emp['Transportation expense'].iloc[10] = np.nan
```

```
C:\Users\Aditya\Anaconda3\lib\site-packages\pandas\core\indexing.py:189: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self._setitem_with_indexer(indexer, value)
```

```
emp['Transportation expense'].iloc[10]
```

```
nan
```

```
#Applying KNN imputation method
emp = pd.DataFrame(KNN(k = 3).fit_transform(emp), columns = emp.columns)
```

```
Imputing row 1/740 with 0 missing, elapsed time: 0.112
Imputing row 101/740 with 1 missing, elapsed time: 0.112
Imputing row 201/740 with 0 missing, elapsed time: 0.112
Imputing row 301/740 with 0 missing, elapsed time: 0.112
Imputing row 401/740 with 0 missing, elapsed time: 0.112
Imputing row 501/740 with 0 missing, elapsed time: 0.112
Imputing row 601/740 with 0 missing, elapsed time: 0.112
Imputing row 701/740 with 0 missing, elapsed time: 0.112
```

```
emp['Transportation expense'].iloc[10]
```

```
259.99999650507766
```

```
#As we have seen that actual value is 260 and knn has given the value of 259.99 which is nearest to the actual value where
#we created the missing value. Hence we will go with KNN method of imputation for missing values.
```

```
#Checking if any missing values
emp.isnull().sum()
```

```
ID                               0
Reason for absence               0
Month of absence                 0
Day of the week                  0
Seasons                          0
Transportation expense           0
Distance from Residence to Work  0
Service time                     0
Age                              0
Work load Average/day            0
Hit target                       0
Disciplinary failure             0
Education                        0
Son                              0
Social drinker                   0
Social smoker                    0
Pet                              0
Weight                           0
Height                           0
Body mass index                  0
Absenteeism time in hours        0
dtype: int64
```

```
#We can  see that now there are no missing values present hence we can move to our next stage of pre processing.
```

Outlier Analysis

```
emp.head(10)
```

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	...	Disciplinary failure	Education	Son	Social drinker	Social smoker	Pet	W
0	11.0	26.0	7.0	3.0	1.0	289.000000	36.0	13.0	33.0	239554.0	...	0.0	1.0	2.0	1.0	0.0	1.0	
1	36.0	0.0	7.0	3.0	1.0	118.000000	13.0	18.0	50.0	239554.0	...	1.0	1.0	1.0	1.0	0.0	0.0	
2	3.0	23.0	7.0	4.0	1.0	179.000000	51.0	18.0	38.0	239554.0	...	0.0	1.0	0.0	1.0	0.0	0.0	
3	7.0	7.0	7.0	5.0	1.0	279.000000	5.0	14.0	39.0	239554.0	...	0.0	1.0	2.0	1.0	1.0	0.0	
4	11.0	23.0	7.0	5.0	1.0	289.000000	36.0	13.0	33.0	239554.0	...	0.0	1.0	2.0	1.0	0.0	1.0	
5	3.0	23.0	7.0	6.0	1.0	179.000000	51.0	18.0	38.0	239554.0	...	0.0	1.0	0.0	1.0	0.0	0.0	
6	10.0	22.0	7.0	6.0	1.0	354.419906	52.0	3.0	28.0	239554.0	...	0.0	1.0	1.0	1.0	0.0	4.0	
7	20.0	23.0	7.0	6.0	1.0	260.000000	50.0	11.0	36.0	239554.0	...	0.0	1.0	4.0	1.0	0.0	0.0	
8	14.0	19.0	7.0	2.0	1.0	155.000000	12.0	14.0	34.0	239554.0	...	0.0	1.0	2.0	1.0	0.0	0.0	
9	1.0	22.0	7.0	2.0	1.0	235.000000	11.0	14.0	37.0	239554.0	...	0.0	3.0	1.0	0.0	0.0	1.0	

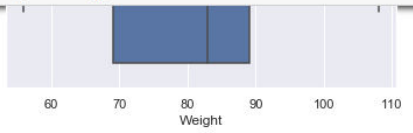
10 rows × 21 columns

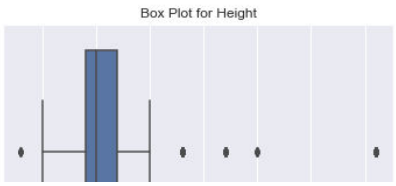
```
emp.shape
```

```
(740, 21)
```

```
#Here we are storing categorical and continuous variables in different objects
variable_num = ['Distance from Residence to Work', 'Service time', 'Age', 'Work load Average/day ', 'Transportation expense',
       'Hit target', 'Weight', 'Height', 'Body mass index','Pet', 'Son', 'Absenteeism time in hours']

variable_cat = ['ID','Reason for absence','Month of absence','Day of the week',
                    'Seasons','Disciplinary failure', 'Education', 'Social drinker',
                    'Social smoker']
```

```
#Draw box plot
for i in variable_num:
    sns.boxplot(emp[i])
    plt.title("Box Plot for "+str(i))
    plt.show()
```






```
#detect and replace outliers with NA
#Extracting quartiles
for i in variable_num:
    q75,q25=np.percentile(emp[i],[75,25])



   ##calculating iqr
    iqr=q75-q25

    #calculating inner and outer fence
    minimum= q25-(iqr*1.5)
    maximum= q75+(iqr*1.5)

    #replace with NA
    emp.loc[emp[i]<minimum,i] = np.nan
    emp.loc[emp[i]>maximum,i] = np.nan
```

```
emp.isnull().sum()
```

```
ID                                0
Reason for absence                0
Month of absence                  0
Day of the week                   0
Seasons                           0
Transportation expense            3
Distance from Residence to Work   0
Service time                      5
Age                               8
Work load Average/day            31
Hit target                       19
Disciplinary failure              0
Education                         0
Son                               0
Social drinker                    0
Social smoker                     0
Pet                              46
Weight                            0
Height                          119
Body mass index                   0
Absenteeism time in hours        46
dtype: int64
```

```
#calculating missing values
missing_val = pd.DataFrame(emp.isnull().sum())
```

```
#Impute with KNN
emp=pd.DataFrame(KNN(k=3).fit_transform(emp), columns = emp.columns)
```

```
Imputing row 1/740 with 0 missing, elapsed time: 0.111
Imputing row 101/740 with 1 missing, elapsed time: 0.111
Imputing row 201/740 with 1 missing, elapsed time: 0.113
Imputing row 301/740 with 0 missing, elapsed time: 0.113
Imputing row 401/740 with 0 missing, elapsed time: 0.113
Imputing row 501/740 with 0 missing, elapsed time: 0.117
Imputing row 601/740 with 0 missing, elapsed time: 0.117
Imputing row 701/740 with 0 missing, elapsed time: 0.117
```

```
emp.isnull().sum()
```

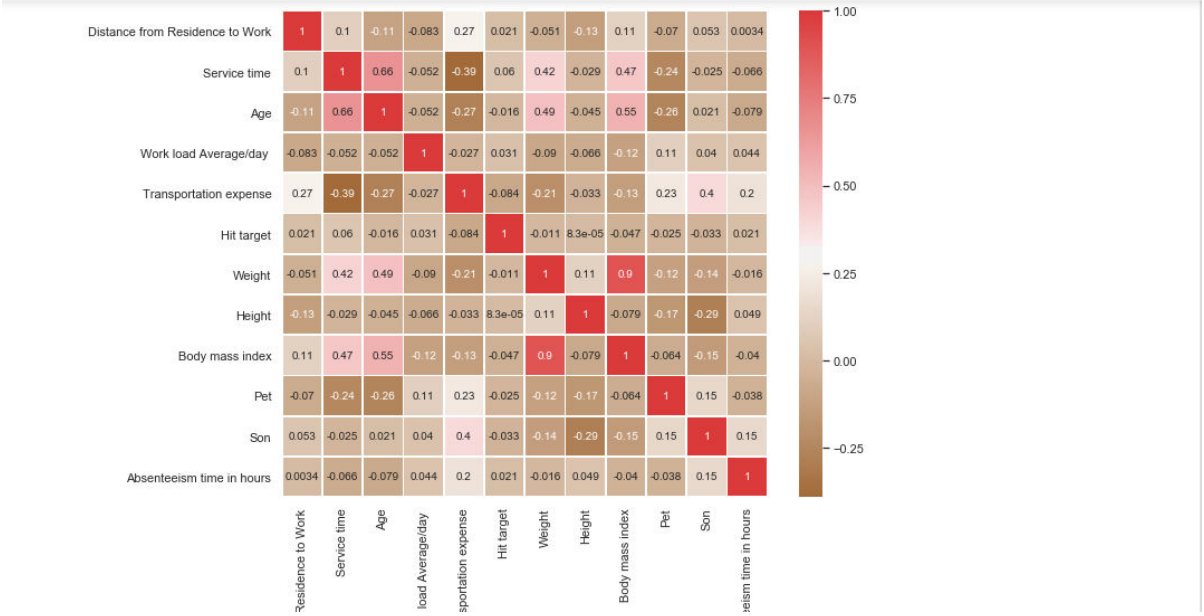
```
ID                                0
Reason for absence                0
Month of absence                  0
Day of the week                   0
Seasons                           0
Transportation expense            0
Distance from Residence to Work   0
Service time                      0
Age                               0
Work load Average/day             0
Hit target                        0
Disciplinary failure              0
Education                         0
Son                               0
Social drinker                    0
Social smoker                     0
Pet                               0
Pet                               0
Weight                            0
Height                            0
Body mass index                   0
Absenteeism time in hours         0
dtype: int64
```

```
#Draw box plot
for i in variable_num:
    sns.boxplot(emp[i])
    plt.title("Box Plot for "+str(i))
    plt.show()
```

Feature Selection

```
br_corr = emp.loc[:,variable_num]
```

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

corr_matrix = br_corr.corr()

sns.heatmap(corr_matrix, mask = np.zeros_like(corr_matrix, dtype=np.bool), cmap = sns.diverging_palette(400,12, as_cmap= True),
            annot = True , linewidths = 0.9,square = True, ax=ax)
```




```
#Loop for Anova as we have categorical and continous varibale
for i in variable_cat:
    f, p = stats.f_oneway(emp[i], emp["Absenteeism time in hours"])
    print("P value for variable "+str(i)+" is "+str(p))
```

```
P value for variable ID is 1.7785014075344148e-172
P value for variable Reason for absence is 3.6736405592138233e-274
P value for variable Month of absence is 8.947828064387552e-27
P value for variable Day of the week is 0.0004917677652965601
P value for variable Seasons is 3.2732455306427215e-42
P value for variable Disciplinary failure is 9.750833449882069e-194
P value for variable Education is 1.3331381651935513e-110
P value for variable Social drinker is 1.135013192977773e-157
P value for variable Social smoker is 3.550465245382907e-192
```

```
#here we are dropping redundant variable.## Here we are dropping continous variable who have high correlation.
emp_reduced = emp.drop(['Weight','Age'],axis=1)
```

```
emp_reduced.shape
```

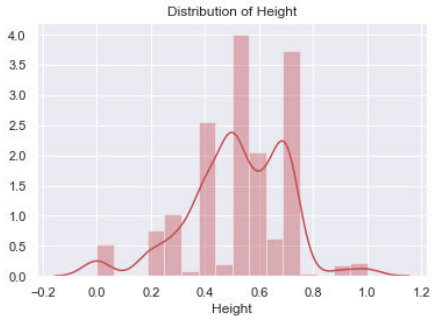
```
(740, 19)
```

```
emp_reduced.head(5)
```

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Work load Average/day	Hit target	Disciplinary failure	Education	Son	Social drinker	Social smoker	Pet	
0	11.0	26.0	7.0	3.0	1.0	289.0	36.0	13.0	239554.0	97.0	0.0	1.0	2.0	1.0	0.0	1.0	172.
1	36.0	0.0	7.0	3.0	1.0	118.0	13.0	18.0	239554.0	97.0	1.0	1.0	1.0	1.0	0.0	0.0	170.
2	3.0	23.0	7.0	4.0	1.0	179.0	51.0	18.0	239554.0	97.0	0.0	1.0	0.0	1.0	0.0	0.0	170.
3	7.0	7.0	7.0	5.0	1.0	279.0	5.0	14.0	239554.0	97.0	0.0	1.0	2.0	1.0	1.0	0.0	168.
4	11.0	23.0	7.0	5.0	1.0	289.0	36.0	13.0	239554.0	97.0	0.0	1.0	2.0	1.0	0.0	1.0	172.

```
#Checking distribution of data
sns.set()
for i in emp_reduced:
    sns.distplot(emp_reduced[i],bins = 'auto', color = 'r')
    plt.title("Distribution of "+str(i))
    plt.show()
```

```
array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



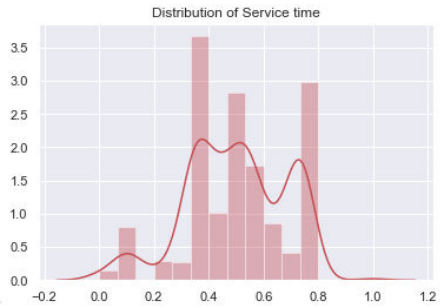

```
#Since we have dropped few variables and stored in emp_reduced. So here we have to updated the variable
#and store them in new object.
variable_num_update = ['Distance from Residence to Work','Pet','Son',
                       'Work load Average/day ', 'Transportation expense',
                       'Hit target', 'Height','Service time',
                       'Body mass index', 'Absenteeism time in hours']

variable_cat_update = ['ID','Reason for absence','Disciplinary failure','Social drinker','Day of the week','Education',
                       'Month of absence','Social smoker','Seasons']
```

```
#Since data is not normally distributed hence we will normalize our data for further analysis
for i in variable_num_update:
    print(i)
    emp_reduced[i] = (emp_reduced[i] - emp_reduced[i].min())/(emp_reduced[i].max()-emp_reduced[i].min())
```

```
Distance from Residence to Work
Pet
Son
Work load Average/day
Transportation expense
Hit target
Height
Service time
Body mass index
Absenteeism time in hours
```

```
#Checking distribution of data
sns.set()
for i in variable_num_update:
    sns.distplot(emp_reduced[i],bins = 'auto', color = 'r')
    plt.title("Distribution of "+str(i))
    plt.show()
```

```
C:\Users\Aditya\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidi
mensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an
array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Creating Dummies

```
#Save target variable first
emp_dummy = pd.DataFrame(emp_reduced['Absenteeism time in hours'])
```

```
emp_dummy.shape
```

```
(740, 1)
```

```
#here in variable_num_update1 we have not added 'Absenteeism time in hours' as we have already added in data frame and adding it
#again will lead to overlap error.
variable_num_update1 = ['Distance from Residence to Work','Pet','Son',
                        'Work load Average/day ', 'Transportation expense',
                        'Hit target', 'Height','Service time',
                        'Body mass index']
```

```
variable_cat_update1 = ['ID','Reason for absence','Disciplinary failure','Social drinker','Day of the week','Education',
                        'Social smoker','Month of absence','Seasons',]
```

```
emp_dummy.head(2)
```

	Absenteeism time in hours
0	0.25
1	0.00

```
#Add continous variables
emp_dummy = emp_dummy.join(emp_reduced[variable_num_update1])
```

```
emp_dummy.shape
```

```
(740, 10)
```

```
##Create dummies for categorical variables. Above we have already stored categorical variable in variable_cat

for i in variable_cat_update1:
    temp = pd.get_dummies(emp_reduced[i], prefix = i)
    emp_dummy = emp_dummy.join(temp)
```

```
emp_dummy.head(5)
```

	Absenteeism time in hours	Distance from Residence to Work	Pet	Son	Work load Average/day	Transportation expense	Hit target	Height	Service time	Body mass index	...	Month of absence_8.0	Month of absence_9.0	Month of absence_10.0	abse
0	0.250	0.659574	0.5	0.50	0.244925	0.657692	0.769231	0.700000	0.521739	0.578947	...	0	0	0	
1	0.000	0.170213	0.0	0.25	0.244925	0.000000	0.769231	0.500001	0.739130	0.631579	...	0	0	0	
2	0.125	0.978723	0.0	0.00	0.244925	0.234615	0.769231	0.500000	0.739130	0.631579	...	0	0	0	
3	0.250	0.000000	0.0	0.50	0.244925	0.619231	0.769231	0.300000	0.565217	0.263158	...	0	0	0	
4	0.125	0.659574	0.5	0.50	0.244925	0.657692	0.769231	0.700000	0.521739	0.578947	...	0	0	0	

5 rows × 118 columns

Dividing data into train and test

```
# Using train_test_split sampling function for test and train data split
X_train, X_test, y_train, y_test = train_test_split(X,emp_dummy, test_size=0.2)
```

Decision Tree

```
# Building model on top of training dataset
fit_DT = DecisionTreeRegressor(max_depth = 5).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)))
```

```
Root Mean Squared Error For Training data = 0.16453802703408338
Root Mean Squared Error For Test data = 0.1754563740288236
R^2 Score(coefficient of determination) = 0.2101739150543113
```

Random Forest ¶

```
#Random Forest
RF_model = RandomForestRegressor(n_estimators = 500, oob_score = True, n_jobs = -1,random_state =50,max_features = "auto",
                                 min_samples_leaf = 50).fit(X_train, y_train)
```

```
# Calculating RMSE for training data to check for over fitting
pred_train = RF_model.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 = RF_model.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)))
```

```
Root Mean Squared Error For Training data = 0.1917232880631965
Root Mean Squared Error For Test data = 0.19499195318494775
R^2 Score(coefficient of determination) = 0.10750282914995798
```

Linear Regression

```
# 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)))
```

```
Root Mean Squared Error For Training data = 0.013022927000721968
Root Mean Squared Error For Test data = 28891.076441671376
R^2 Score(coefficient of determination) = -17444084547.83734
```

Dimensionality reduction using PCA

```
# Dimensionality reduction by using pca
emp1 = emp_dummy['Absenteeism time in hours']
```

```
#emp_dummy.drop(['Absenteeism time in hours'], inplace = True, axis=1)
emp_dummy.shape
```

```
(740, 118)
```

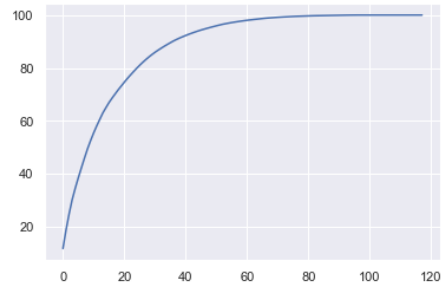
```
from sklearn.decomposition import PCA

# Converting data to numpy array
X = emp_dummy.values
```

```
# Data has 118 variables so no of components of PCA = 118
pca = PCA(n_components=118)
pca.fit(X)
```

```
PCA(copy=True, iterated_power='auto', n_components=118, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

```
# The amount of variance that each PC explains
var= pca.explained_variance_ratio_
```

```
# Cumulative Variance explains
sns.set()
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
plt.plot(var1)
plt.show()
```




```
# From the above plot selecting 50 components since it explains almost 95+ % data variance
pca = PCA(n_components=50)



# Fitting the selected components to the data
pca.fit(X)
```

Dividing data into train

```
# Using train_test_split sampling function for test and train data split
X_train, X_test, y_train, y_test = train_test_split(X,emp1, test_size=0.2)
```

Decision Tree

```
# Building model on top of training dataset
fit_DT = DecisionTreeRegressor(max_depth = 5).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)))
```

```
Root Mean Squared Error For Training data = 0.0014547258030807727
Root Mean Squared Error For Test data = 0.009282941496556348
R^2 Score(coefficient of determination) = 0.9984520572905576
```

Random Forest

```
#Random Forest
RF_model = RandomForestRegressor(n_estimators = 500, oob_score = True, n_jobs = -1,random_state =50,max_features = "auto",
                                 min_samples_leaf = 50).fit(X_train, y_train)
```

```
# Calculating RMSE for training data to check for over fitting
pred_train = RF_model.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 = RF_model.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)))
```

```
Root Mean Squared Error For Training data = 0.08463919371751547
Root Mean Squared Error For Test data = 0.07135560349067667
R^2 Score(coefficient of determination) = 0.8598614268421698
```

Linear Regression

```
# 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)))
```

```
Root Mean Squared Error For Training data = 4.529890363154557e-16
Root Mean Squared Error For Test data = 0.00030505994938045586
R^2 Score(coefficient of determination) = 0.9999979137208953
```

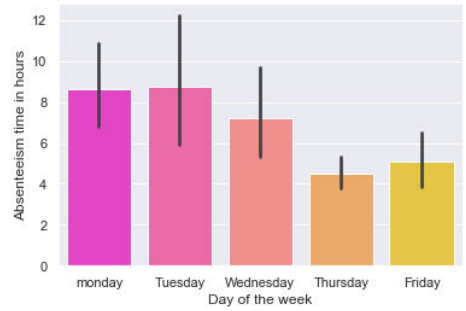
```
#After applying PCA and doing dimesion reduction we can clearly see that Linear regression performs better as compared to
#Random forest and decision tree. So we will go for linear regression model.
```

```
emp=df.copy()
```

```
emp['Day of the week'] = emp['Day of the week'].replace(2, "monday")
emp['Day of the week'] = emp['Day of the week'].replace(3, "Tuesday")
emp['Day of the week'] = emp['Day of the week'].replace(4, "Wednesday")
emp['Day of the week'] = emp['Day of the week'].replace(5, "Thursday")
emp['Day of the week'] = emp['Day of the week'].replace(6, "Friday")
```

```
sns.barplot(x='Day of the week', y='Absenteeism time in hours', palette = 'spring', data = emp, order  =['monday','Tuesday',
                                                                                                          'Wednesday','Thursday','
```

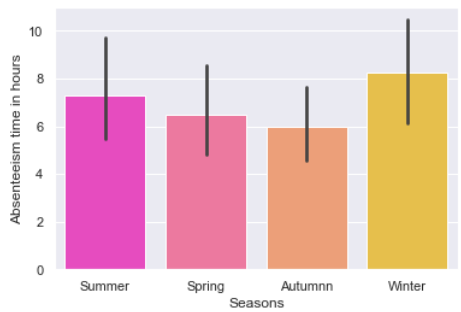
```
<matplotlib.axes._subplots.AxesSubplot at 0x23d90d06ac8>
```




```
emp['Seasons'] = emp['Seasons'].replace(1, "Summer")
emp['Seasons'] = emp['Seasons'].replace(2, "Autumnn")
emp['Seasons'] = emp['Seasons'].replace(3, "Winter")
emp['Seasons'] = emp['Seasons'].replace(4, "Spring")
```

```
sns.barplot(x='Seasons', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

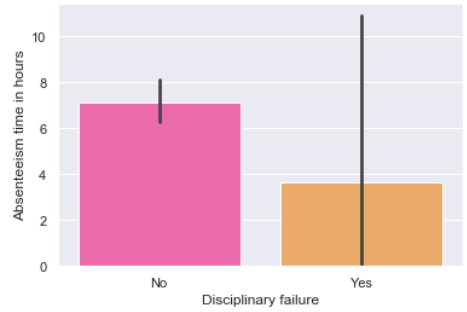
```
<matplotlib.axes._subplots.AxesSubplot at 0x23d90f69128>
```




```
emp['Disciplinary failure'] = emp['Disciplinary failure'].replace(0, "No")
emp['Disciplinary failure'] = emp['Disciplinary failure'].replace(1, "Yes")
```

```
sns.barplot(x='Disciplinary failure', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

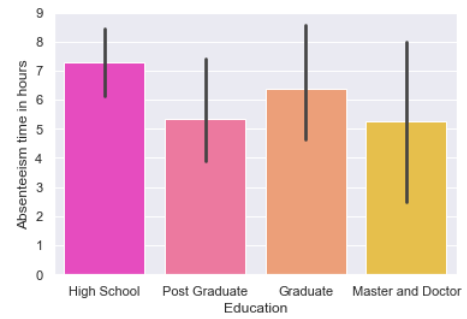
```
<matplotlib.axes._subplots.AxesSubplot at 0x23d90fe6710>
```




```
emp['Education'] = emp['Education'].replace(1, "High School")
emp['Education'] = emp['Education'].replace(2, "Graduate")
emp['Education'] = emp['Education'].replace(3, "Post Graduate")
emp['Education'] = emp['Education'].replace(4, "Master and Doctor")
```

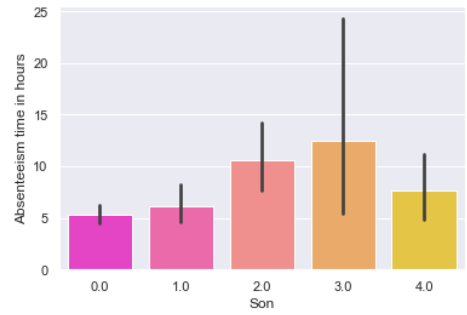
```
sns.barplot(x='Education', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x23d91022358>
```




```
sns.barplot(x='Son', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

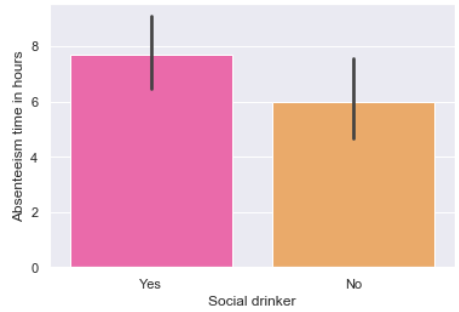
```
<matplotlib.axes._subplots.AxesSubplot at 0x23d9108de48>
```




```
emp['Social drinker'] = emp['Social drinker'].replace(0, "No")
emp['Social drinker'] = emp['Social drinker'].replace(1, "Yes")
```

```
sns.barplot(x='Social drinker', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

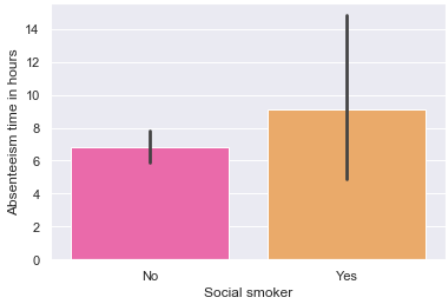
```
<matplotlib.axes._subplots.AxesSubplot at 0x23d91102240>
```




```
emp['Social smoker'] = emp['Social smoker'].replace(0, "No")
emp['Social smoker'] = emp['Social smoker'].replace(1, "Yes")
```

```
sns.barplot(x='Social smoker', y='Absenteeism time in hours', palette = 'spring',  data = emp)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x23d9115b470>
```



References

1. For Data pre-processing and Model Development - https://learning.edwisor.com/

2. For PCA - https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/

   https://www.youtube.com/watch?v=_UVHneBUBW0

3. For Visualization – https://www.youtube.com/channel/UCQTQ0AbOupKNxKKY-_x46OQ

   https://learning.edwisor.com/