Employee Absenteeism Project

The main agenda of this project is to reduce the number of abseenteism in XYZ courier company.

In [1]:
```
#import the necessary libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sbn
%matplotlib inline
from fancyimpute import KNN
from scipy import stats
from sklearn.metrics import r2_score
```

```
Using TensorFlow backend.
```

Understanding the data

To know the basics of the data - shape, data types of the variables and etc.,

In [2]:
```
#To set the working directory and cross checking it
os.chdir("E:\edWisor\Assignments & Solutions\Project_Emp")
os.getcwd()
```

Out[2]:
```
'E:\\edWisor\\Assignments & Solutions\\Project_Emp'
```

In [3]:
```
#Loading the dataset
emp_data = pd.read_excel('Absenteeism_at_work_Project.xls')
```

In [4]:
```
#To check the top rows
emp_data.head()
```

Out[4]:

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work loa Average/day
0	11	26.0	7.0	3	1	289.0	36.0	13.0	33.0	239554.(
1	36	0.0	7.0	3	1	118.0	13.0	18.0	50.0	239554.(
2	3	23.0	7.0	4	1	179.0	51.0	18.0	38.0	239554.(
3	7	7.0	7.0	5	1	279.0	5.0	14.0	39.0	239554.(
4	11	23.0	7.0	5	1	289.0	36.0	13.0	33.0	239554.(

5 rows × 21 columns

In [5]:
```
##To check the dimensions of our data
emp_data.shape
```

Out[5]:
```
(740, 21)
```

In [6]:
```
#To check the data types
emp_data.dtypes
```

Out[6]:
```
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
```

In [7]:
```
#To check the number of unique values in each variable in our dataset
emp_data.nunique()
```

Out[7]:
```
ID                              36
Reason for absence              28
Month of absence                13
Day of the week                  5
Seasons                          4
Transportation expense          24
Distance from Residence to Work 25
Service time                    18
Age                             22
Work load Average/day           38
Hit target                      13
Disciplinary failure             2
Education                        4
Son                              5
Social drinker                   2
Social smoker                    2
Pet                              6
Weight                          26
Height                          14
Body mass index                 17
Absenteeism time in hours       19
dtype: int64
```

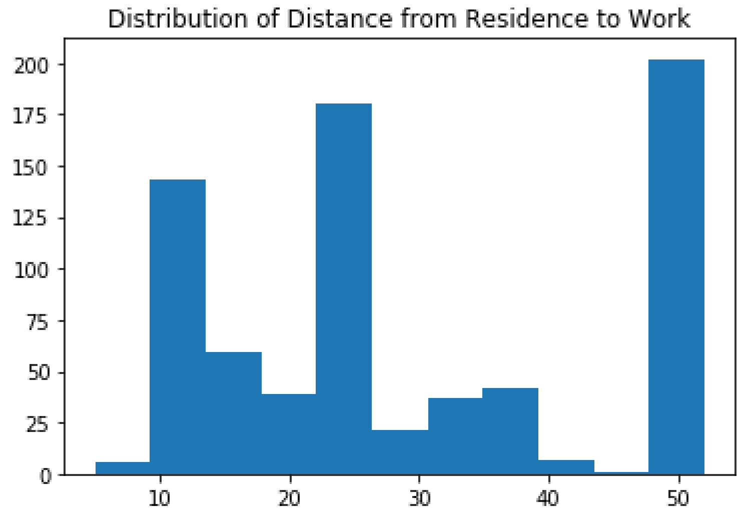
In [8]:
```
#Transforming the data types appropriately
emp_data['ID'] = emp_data['ID'].astype('category')
emp_data['Reason for absence'] = emp_data['Reason for absence'].astype('catego
ry')
emp_data['Month of absence'] = emp_data['Month of absence'].replace(0,np.nan)
emp_data['Month of absence'] = emp_data['Month of absence'].astype('category')
emp_data['Day of the week'] = emp_data['Day of the week'].astype('category')
emp_data['Seasons'] = emp_data['Seasons'].astype('category')
emp_data['Disciplinary failure'] = emp_data['Disciplinary failure'].astype('ca
tegory')
emp_data['Education'] = emp_data['Education'].astype('category')
emp_data['Son'] = emp_data['Son'].astype('category')
emp_data['Social drinker'] = emp_data['Social drinker'].astype('category')
emp_data['Social smoker'] = emp_data['Social smoker'].astype('category')
emp_data['Pet'] = emp_data['Pet'].astype('category')
```

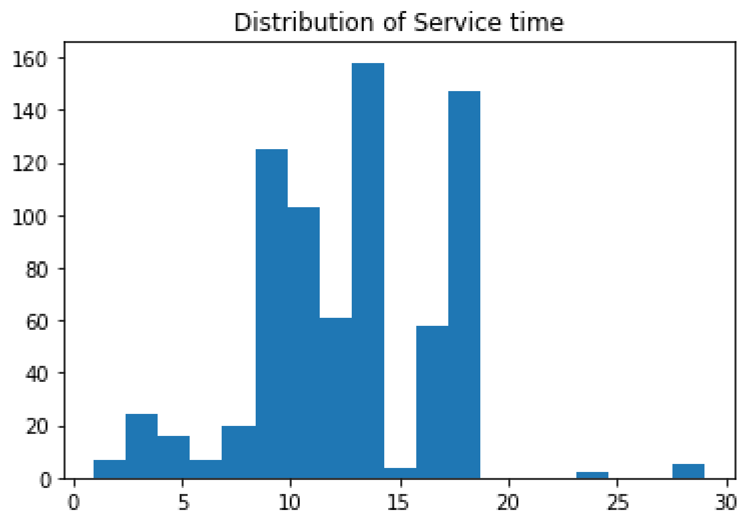
In [9]:
```
#Categorising the variables according to their data types
cont_var = ['Distance from Residence to Work', 'Service time', 'Age', 'Work lo
ad Average/day ', 'Transportation expense',
       'Hit target', 'Weight', 'Height', 'Body mass index', 'Absenteeism time
 in hours']

cat_var = ['ID','Reason for absence','Month of absence','Day of the week',
                     'Seasons','Disciplinary failure', 'Education', 'Social dr
inker',
                     'Social smoker', 'Pet', 'Son']
```

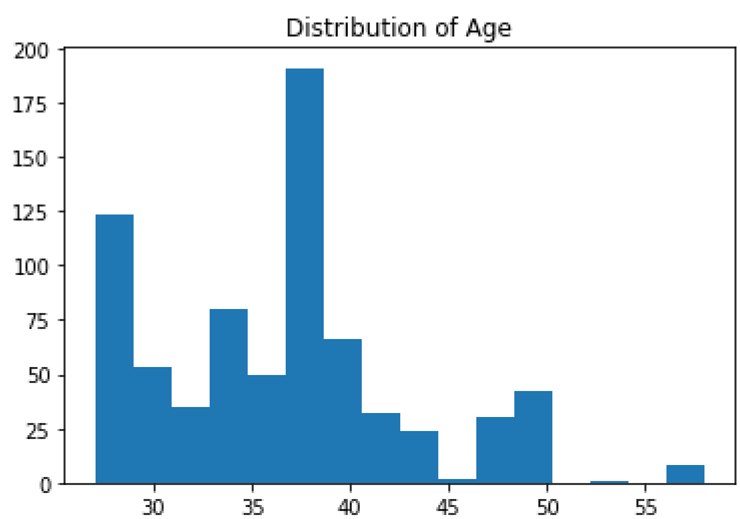
Exploratory data analysis

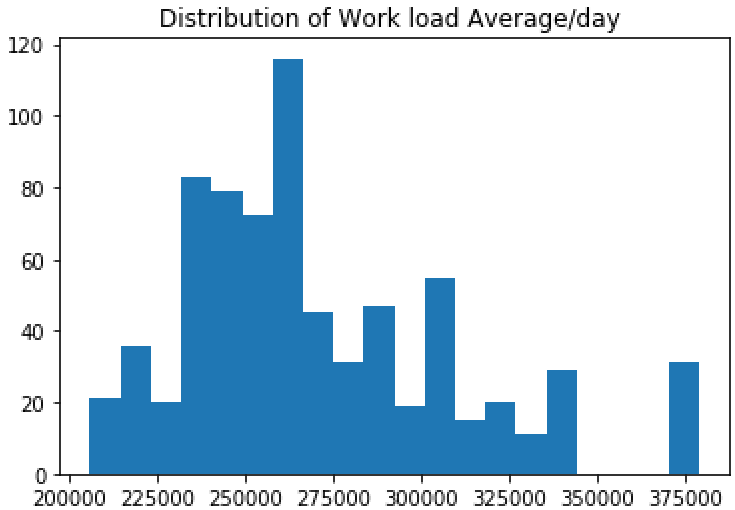
To know the distribution of the data by using the plots and get some basic insights

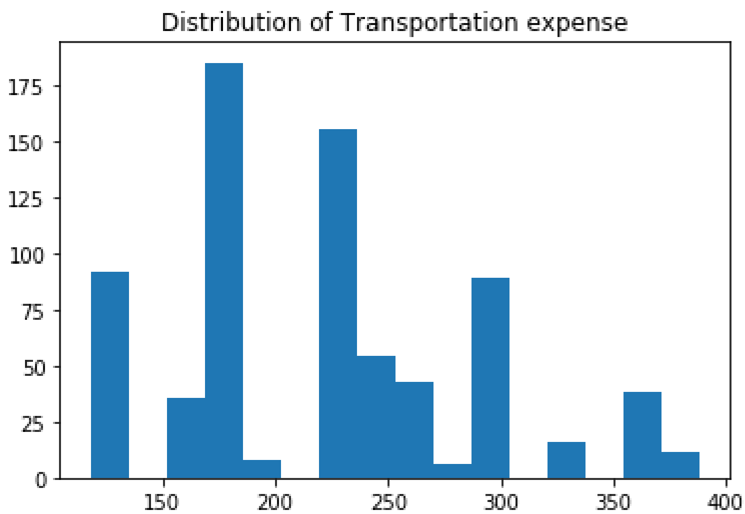
In [10]:
```
#Checking the distribution of the continuous variables
for i in cont_var:
    plt.hist(emp_data[i].dropna(),bins = 'auto')
    plt.title("Distribution of " + str(i))
    plt.show()
```

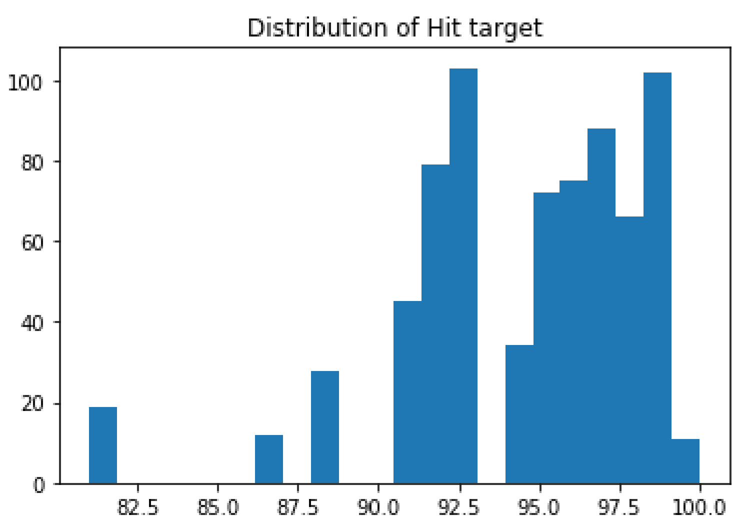

Distribution of Distance from Residence to Work
200
175
150
125
100
75
50
25
0
10
20
30
40
50

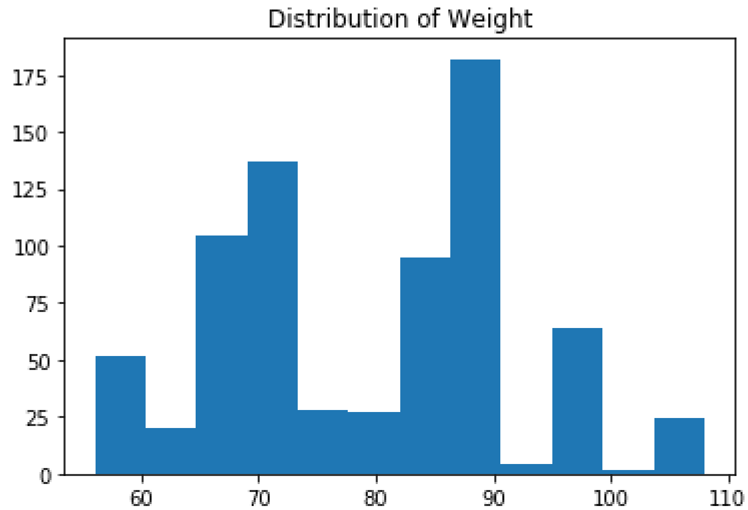

Distribution of Service time
160
140
120
100
80
60
40
20
0
0
5
10
15
20
25
30

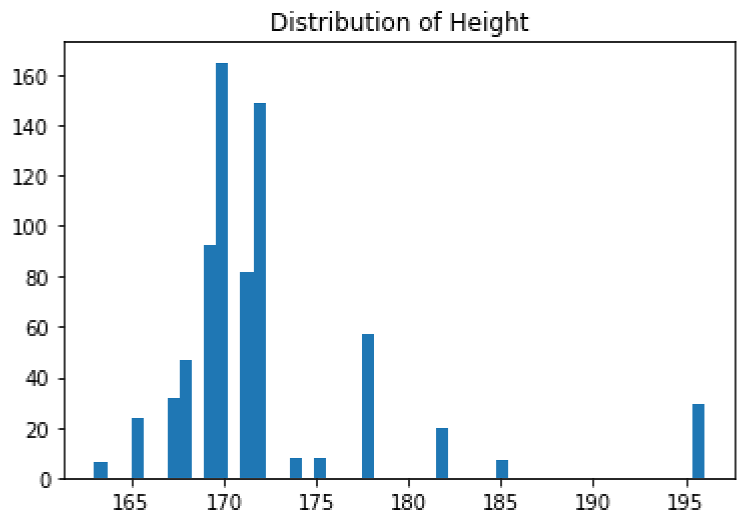

Distribution of Age
200
175
150
125
100
75
50
25
0
30
35
40
45
50
55

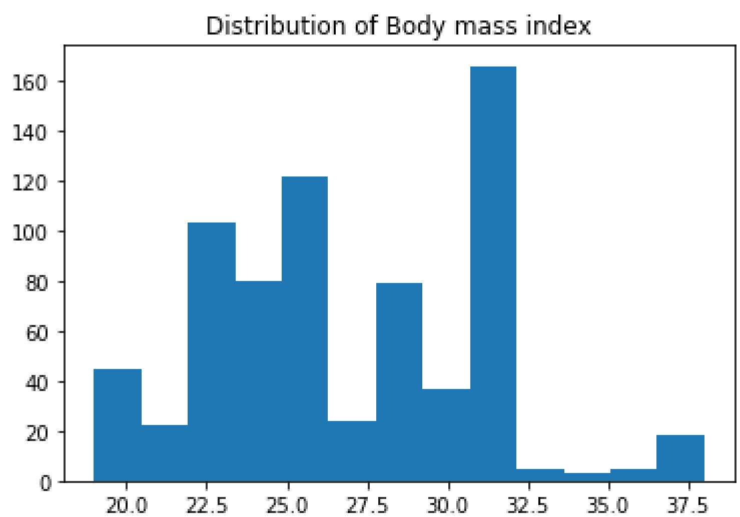

Distribution of Work load Average/day
120
100
80
60
40
20
0
200000
225000
250000
275000
300000
325000
350000
375000

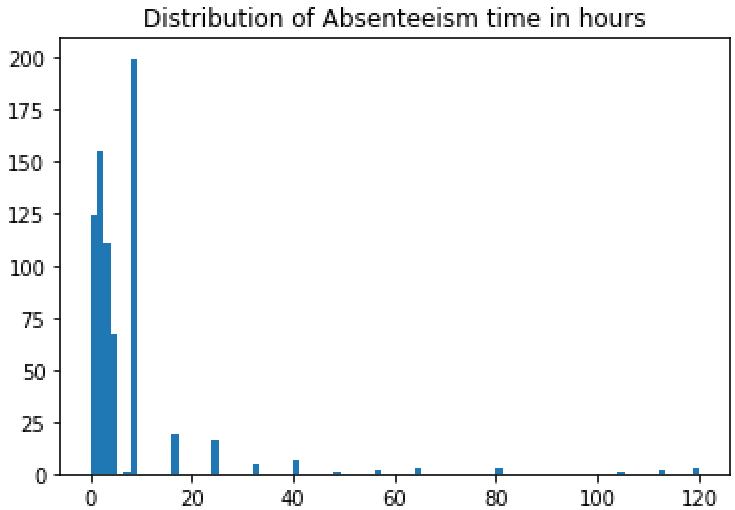

Distribution of Transportation expense
175
150
125
100
75
50
25
0
150
200
250
300
350
400


Distribution of Hit target
100
80
60
40
20
0
82.5
85.0
87.5
90.0
92.5
95.0
97.5
100.0


Distribution of Weight
175
150
125
100
75
50
25
0
60
70
80
90
100
110


Distribution of Height
160
140
120
100
80
60
40
20
0
165
170
175
180
185
190
195


Distribution of Body mass index
160
140
120
100
80
60
40
20
0
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5



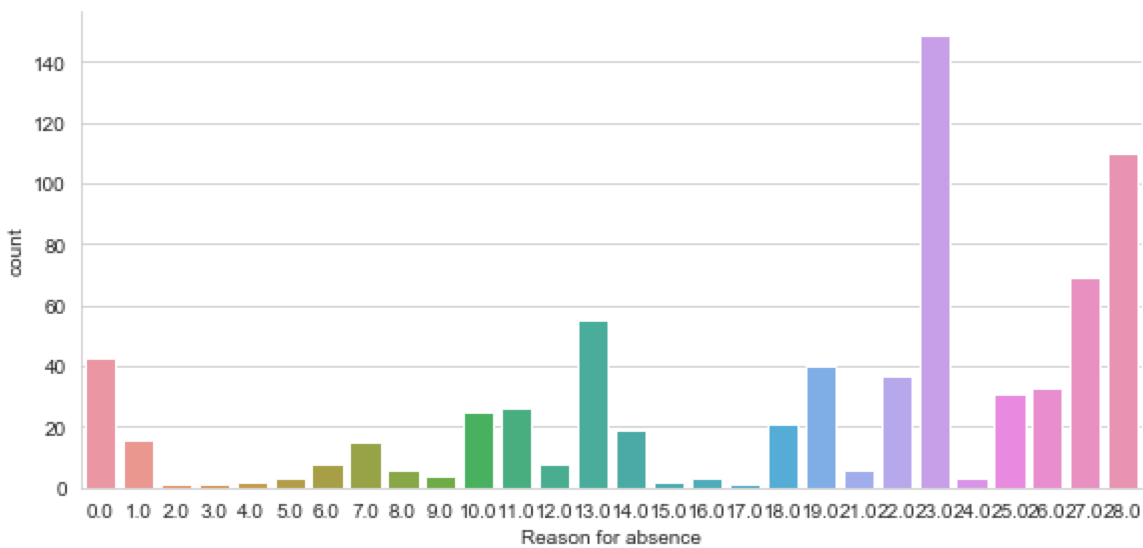
From the above plots w.r.t. continuous variables, it is found that no variable has a normal or uniform distribution.

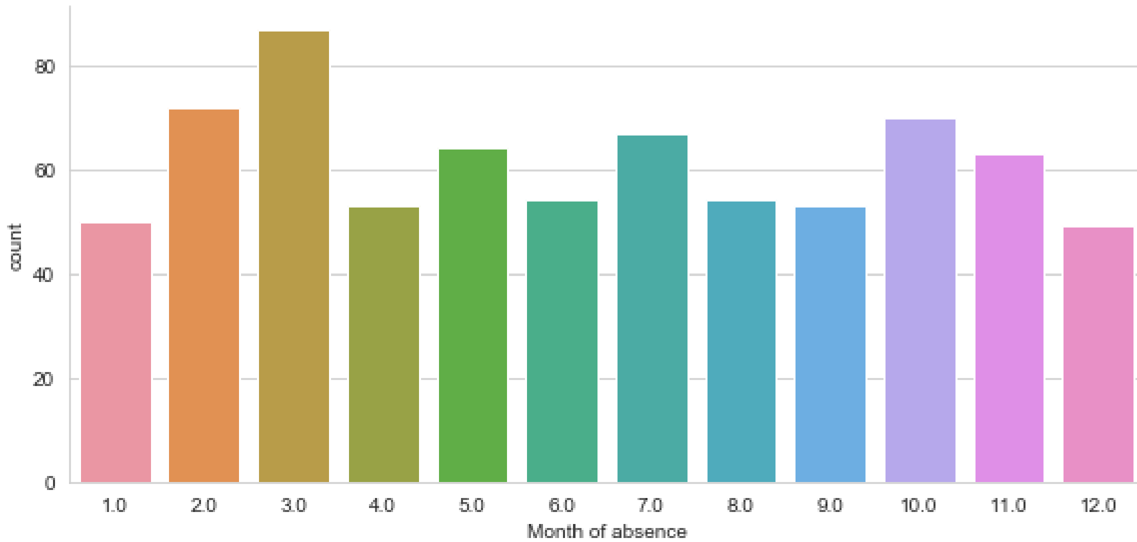
In [11]:
```
#Checking the distribution of categorical variables
sbn.set_style("whitegrid")
sbn.factorplot(data=emp_data, x='Reason for absence', kind= 'count',size=4,asp
ect=2)
sbn.factorplot(data=emp_data, x='Month of absence', kind= 'count',size=4,aspec
t=2)
sbn.factorplot(data=emp_data, x='Seasons', kind= 'count',size=4,aspect=2)
sbn.factorplot(data=emp_data, x='Education', kind= 'count',size=4,aspect=2)
sbn.factorplot(data=emp_data, x='Social drinker', kind= 'count',size=4,aspect=
2)
sbn.factorplot(data=emp_data, x='Social smoker', kind= 'count',size=4,aspect=2
)
sbn.factorplot(data=emp_data, x='ID', kind= 'count',size=4,aspect=2)
sbn.factorplot(data=emp_data, x='Disciplinary failure', kind= 'count',size=4,a
spect=2)
sbn.factorplot(data=emp_data, x='Day of the week', kind= 'count',size=4,aspect
=2)
sbn.factorplot(data=emp_data, x='Pet', kind= 'count',size=4,aspect=2)
sbn.factorplot(data=emp_data, x='Son', kind= 'count',size=4,aspect=2)
```

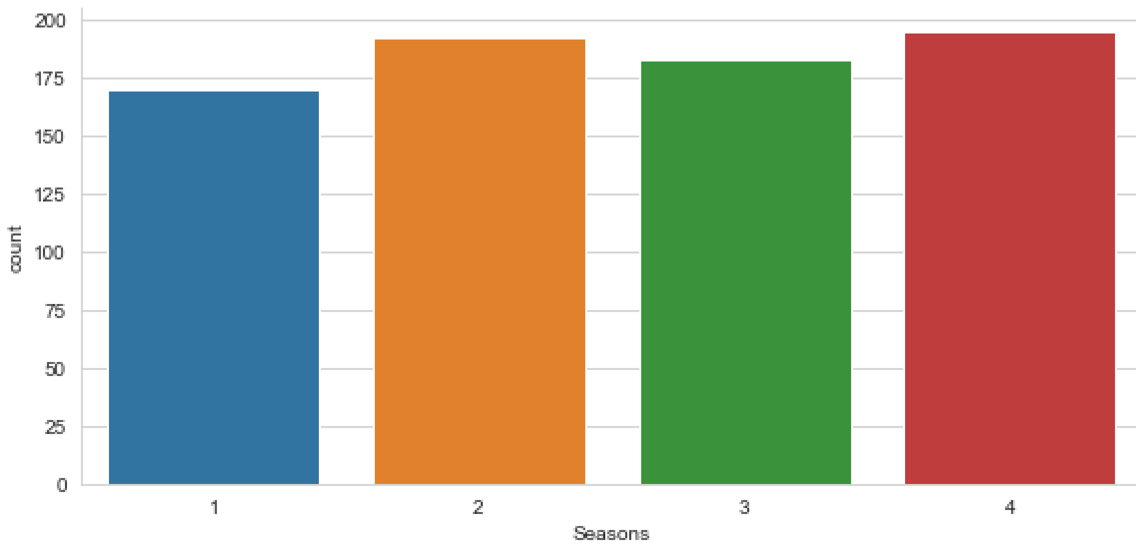
```
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserW
arning: The `factorplot` function has been renamed to `catplot`. The original
name will be removed in a future release. Please update your code. Note that
the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catp
lot`.
  warnings.warn(msg)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserW
arning: The `size` paramter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserW
arning: The `factorplot` function has been renamed to `catplot`. The original
name will be removed in a future release. Please update your code. Note that
the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catp
lot`.
  warnings.warn(msg)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserW
arning: The `size` paramter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserW
arning: The `factorplot` function has been renamed to `catplot`. The original
name will be removed in a future release. Please update your code. Note that
the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catp
lot`.
  warnings.warn(msg)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserW
arning: The `size` paramter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserW
arning: The `factorplot` function has been renamed to `catplot`. The original
name will be removed in a future release. Please update your code. Note that
the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catp
lot`.
  warnings.warn(msg)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserW
arning: The `size` paramter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserW
arning: The `factorplot` function has been renamed to `catplot`. The original
name will be removed in a future release. Please update your code. Note that
the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catp
lot`.
  warnings.warn(msg)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserW
arning: The `size` paramter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserW
arning: The `factorplot` function has been renamed to `catplot`. The original
name will be removed in a future release. Please update your code. Note that
the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catp
lot`.
  warnings.warn(msg)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserW
```

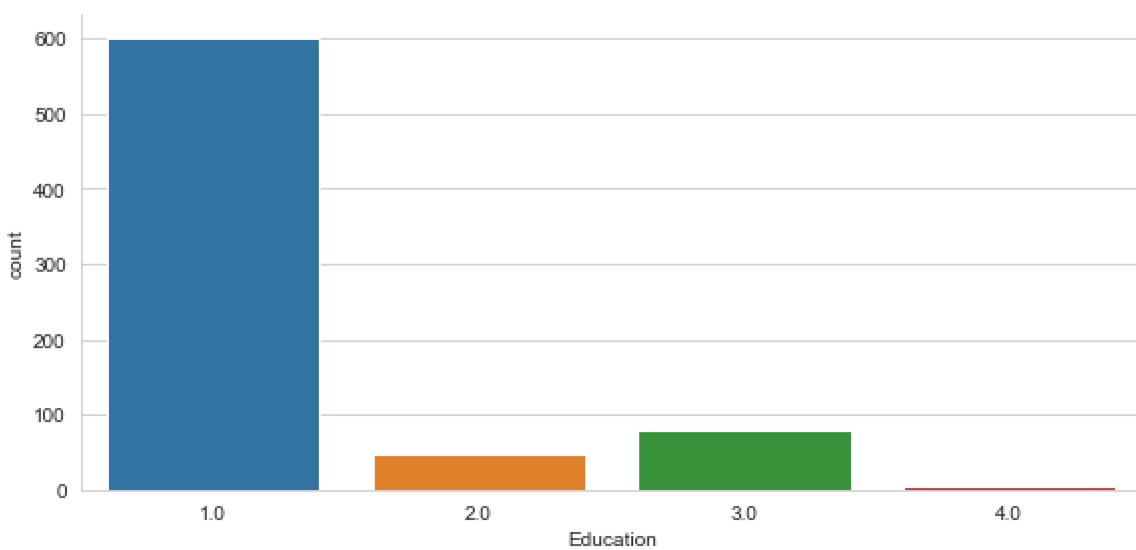
```
arning: The `size` paramter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserW
arning: The `factorplot` function has been renamed to `catplot`. The original
name will be removed in a future release. Please update your code. Note that
the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catp
lot`.
  warnings.warn(msg)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserW
arning: The `size` paramter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserW
arning: The `factorplot` function has been renamed to `catplot`. The original
name will be removed in a future release. Please update your code. Note that
the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catp
lot`.
  warnings.warn(msg)
C:\Users\abhis\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserW
arning: The `size` paramter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
```

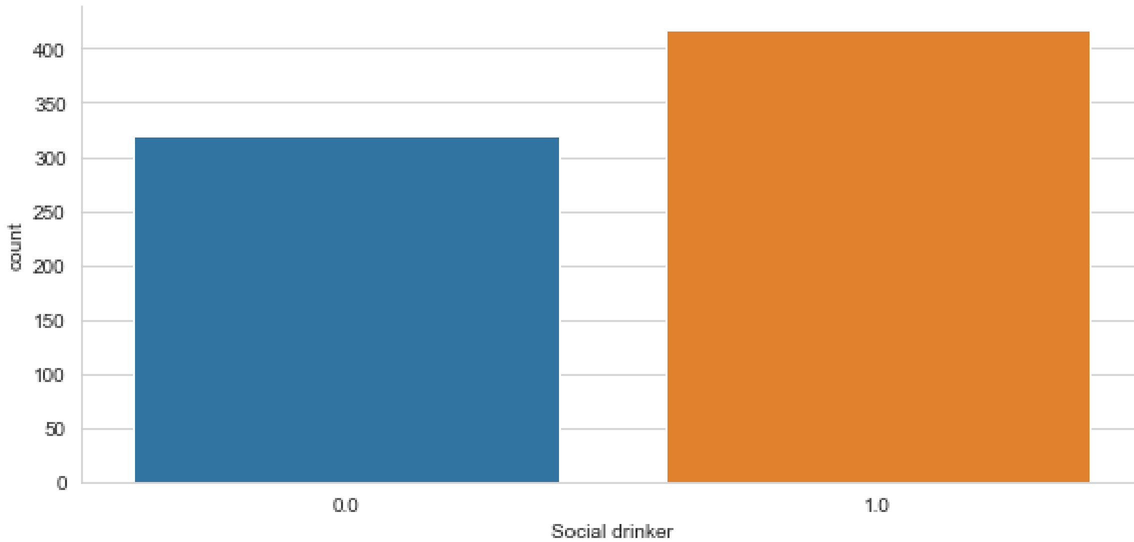
Out[11]: <seaborn.axisgrid.FacetGrid at 0x25cc82c98d0>

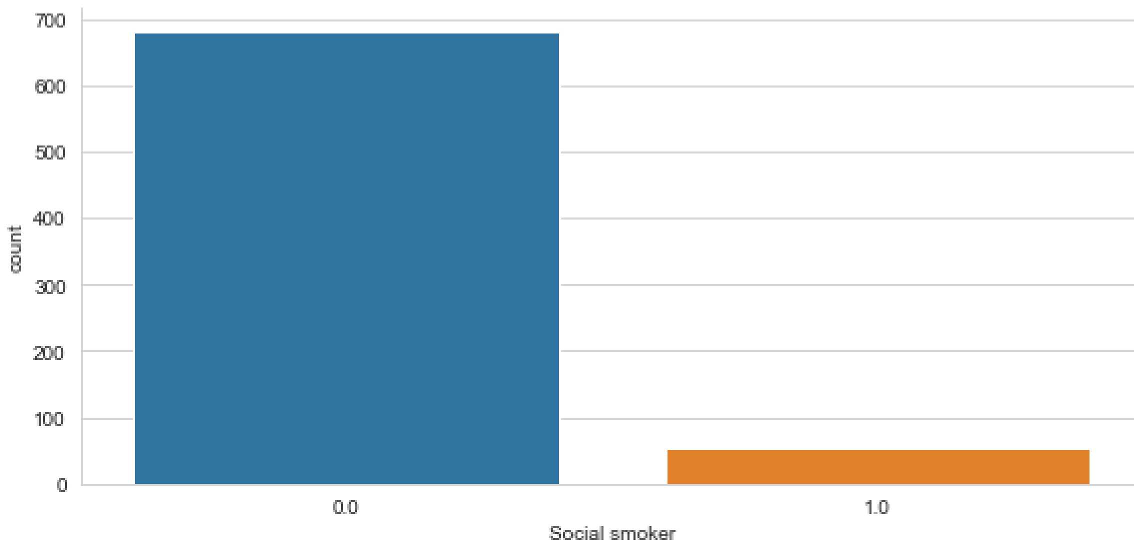


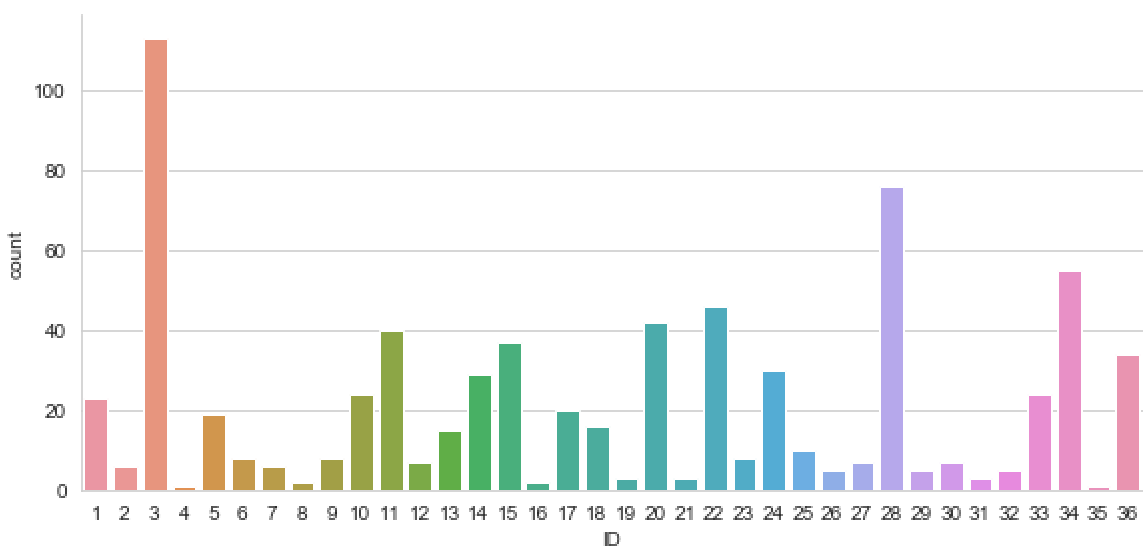

count
Month of absence
0
20
40
60
80
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0

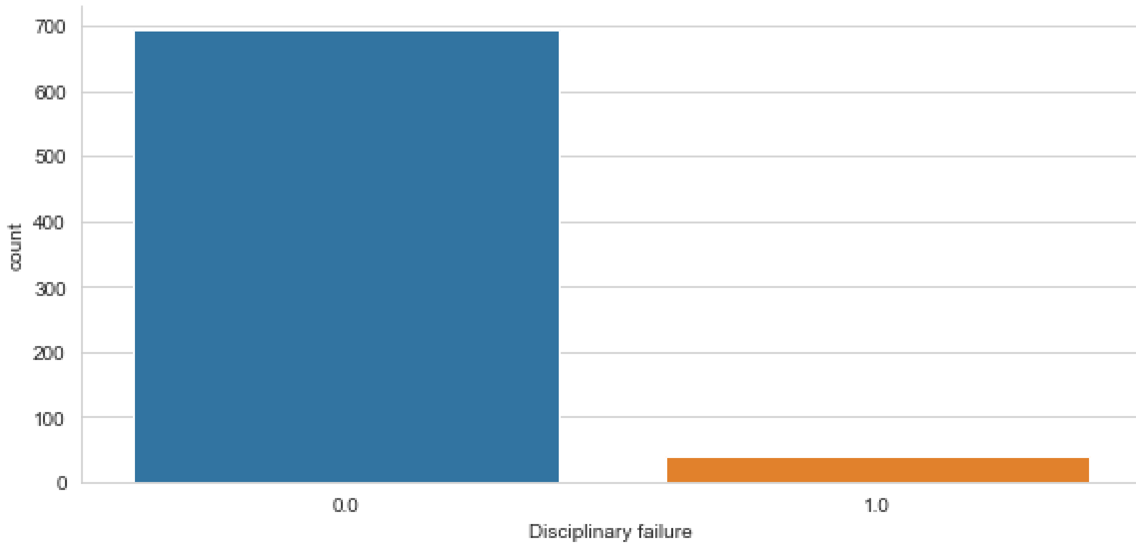

count
Seasons
0
25
50
75
100
125
150
175
200
1
2
3
4

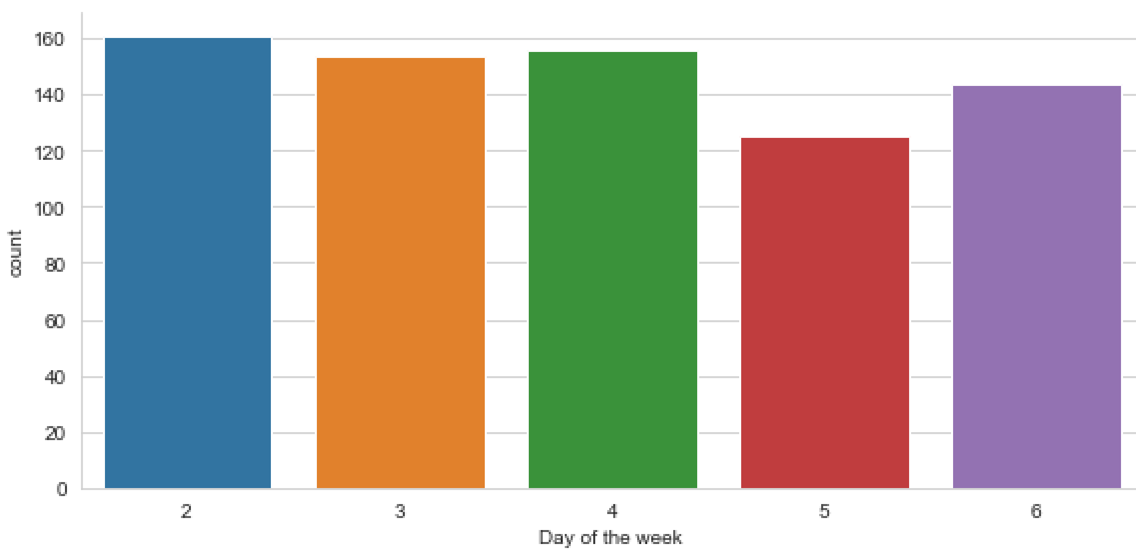

count
Education
0
100
200
300
400
500
600
1.0
2.0
3.0
4.0

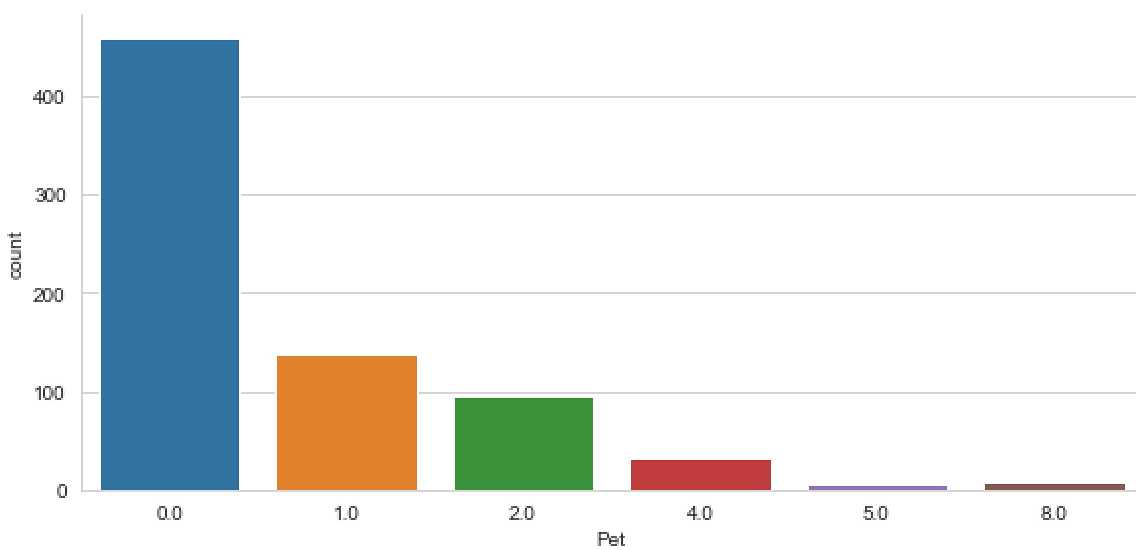

count
0
50
100
150
200
250
300
350
400
0.0
1.0
Social drinker

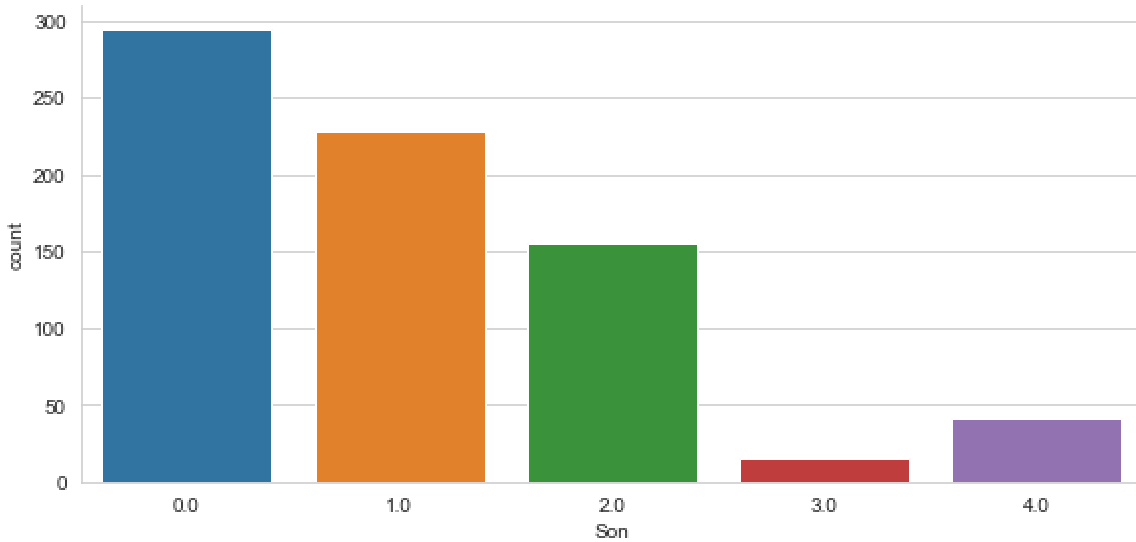

count
0
100
200
300
400
500
600
700
0.0
1.0
Social smoker


count
0
20
40
60
80
100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
ID


count
700
600
500
400
300
200
100
0
0.0
1.0
Disciplinary failure


count
160
140
120
100
80
60
40
20
0
2
3
4
5
6
Day of the week

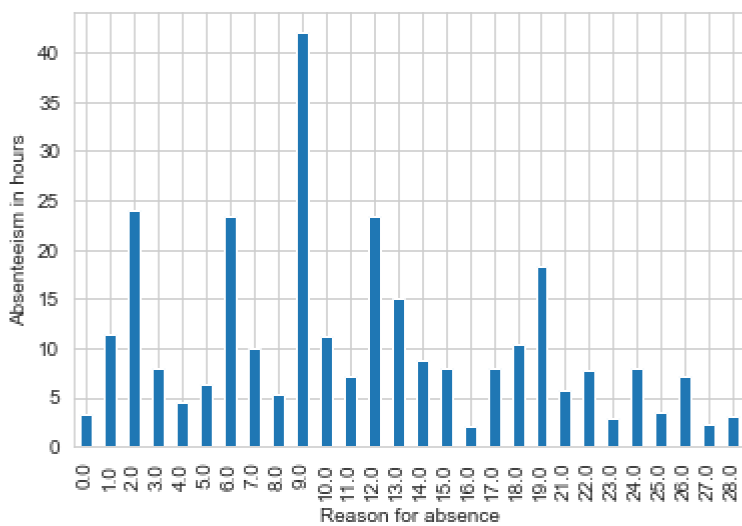

count
400
300
200
100
0
0.0
1.0
2.0
4.0
5.0
8.0
Pet



Let's group the data with each categorical variable and find the mean of `Absenteeism time in hours` for each category. For the categorical variable `ID` , we will consider the sum of the `Absenteeism time in hours` so that we can find the `ID` with the maximum number of absenteeism.

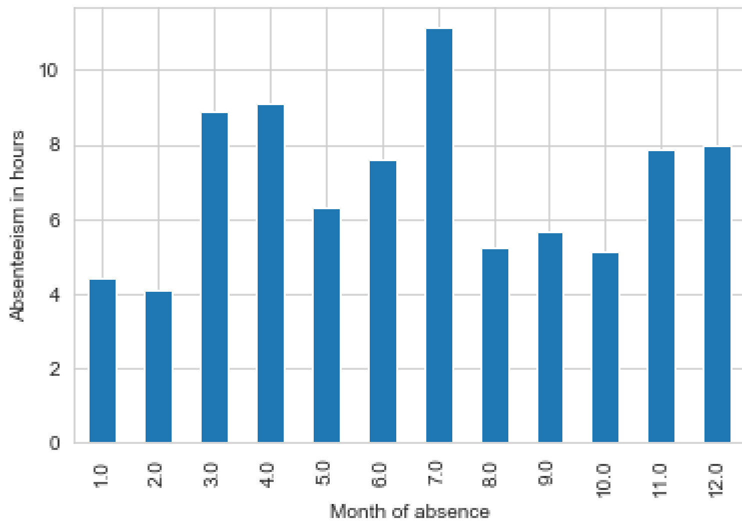
In [12]:
```
# Grouping the data using Reason for absence against our target variable and p
lotting bar plot
emp_data.groupby('Reason for absence').mean()['Absenteeism time in hours'].plo
t.bar()
plt.ylabel('Absenteeism in hours')
```

Out[12]:
```
Text(0, 0.5, 'Absenteeism in hours')
```



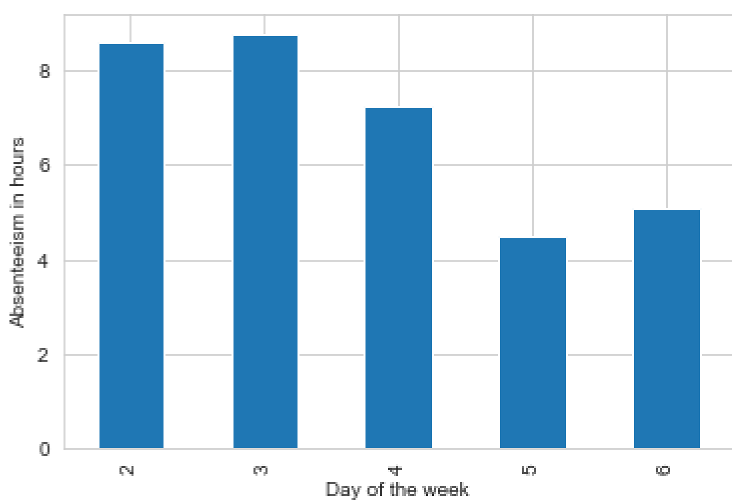
In [13]:
```
# Grouping the data using Month of absence against our target variable and plo
tting bar plot
emp_data.groupby('Month of absence').mean()['Absenteeism time in hours'].plot.
bar()
plt.ylabel('Absenteeism in hours')
```

Out[13]: Text(0, 0.5, 'Absenteeism in hours')



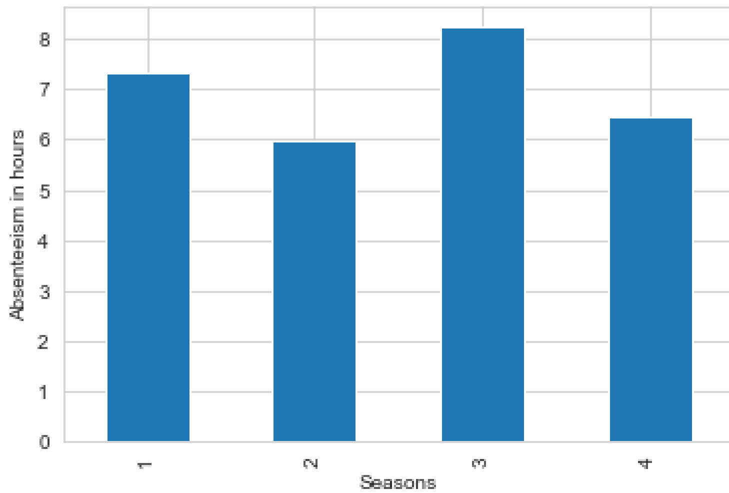
In [14]:
```
# Grouping the data using Day of the week against our target variable and plot
ting bar plot
emp_data.groupby(['Day of the week']).mean()['Absenteeism time in hours'].plot
.bar()
plt.ylabel('Absenteeism in hours')
```

Out[14]: Text(0, 0.5, 'Absenteeism in hours')



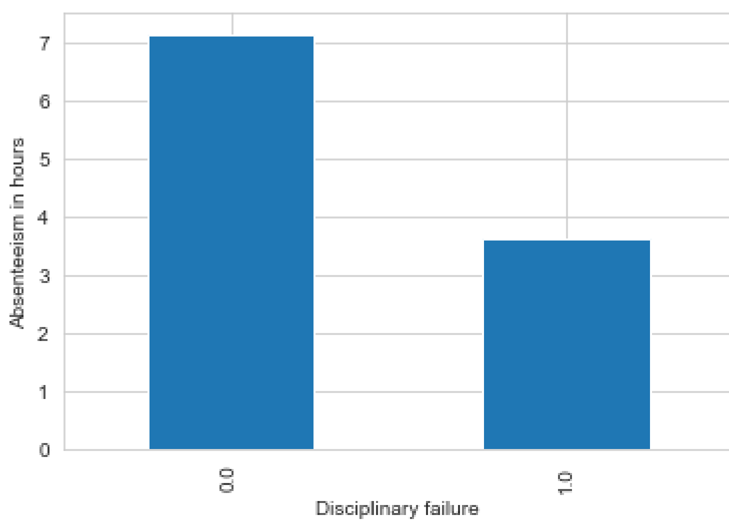
In [15]:
```
# Grouping the data using Seasons against our target variable and plotting bar plot
emp_data.groupby(['Seasons']).mean()['Absenteeism time in hours'].plot.bar()
plt.ylabel('Absenteeism in hours')
```

Out[15]: Text(0, 0.5, 'Absenteeism in hours')



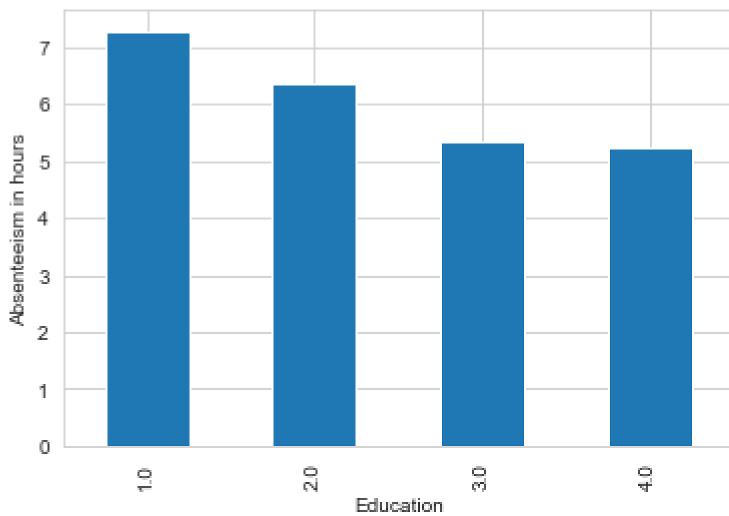
In [16]:
```
# Grouping the data using Disciplinary failure against our target variable and plotting bar plot
emp_data.groupby('Disciplinary failure').mean()['Absenteeism time in hours'].plot.bar()
plt.ylabel('Absenteeism in hours')
```

Out[16]: Text(0, 0.5, 'Absenteeism in hours')



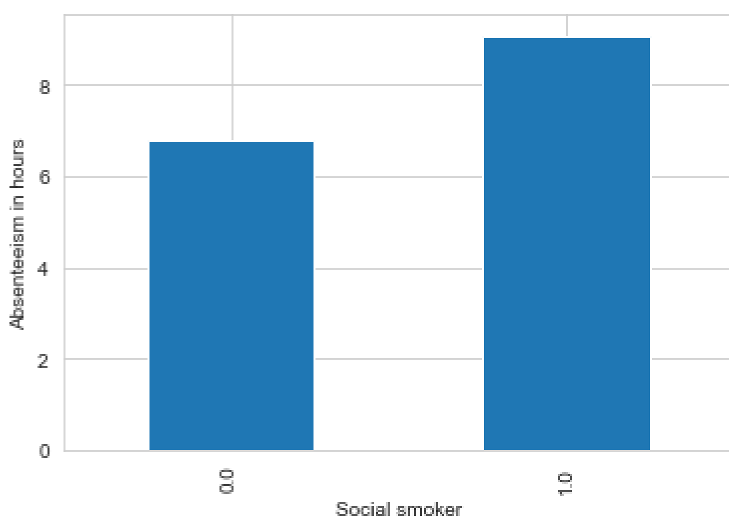
In [17]:
```
# Grouping the data using Education against our target variable and plotting b
ar plot
emp_data.groupby('Education').mean()['Absenteeism time in hours'].plot.bar()
plt.ylabel('Absenteeism in hours')
```

Out[17]: Text(0, 0.5, 'Absenteeism in hours')



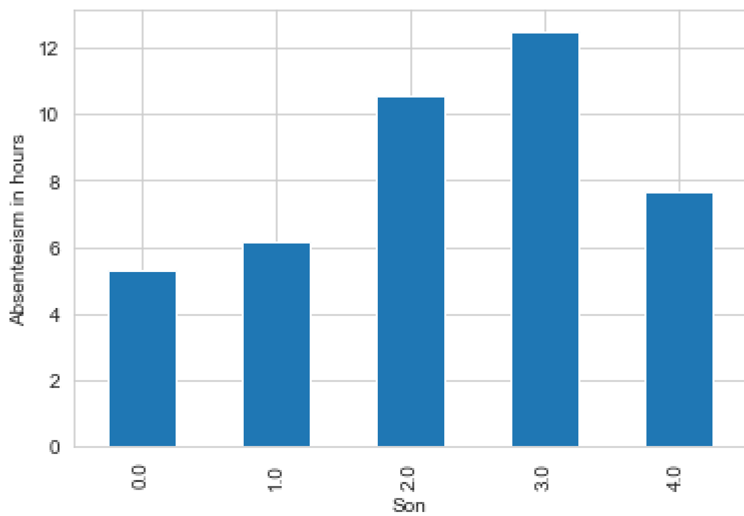
In [18]:
```
# Grouping the data using Social smoker against our target variable and plotti
ng bar plot
emp_data.groupby('Social smoker').mean()['Absenteeism time in hours'].plot.bar
()
plt.ylabel('Absenteeism in hours')
```

Out[18]: Text(0, 0.5, 'Absenteeism in hours')



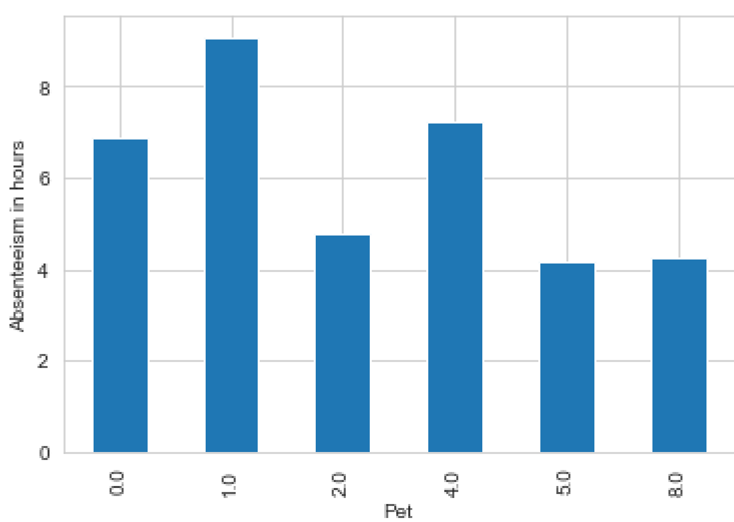
In [19]:
```
# Grouping the data using Son against our target variable and plotting bar plo
t
emp_data.groupby('Son').mean()['Absenteeism time in hours'].plot.bar()
plt.ylabel('Absenteeism in hours')
```

Out[19]: Text(0, 0.5, 'Absenteeism in hours')



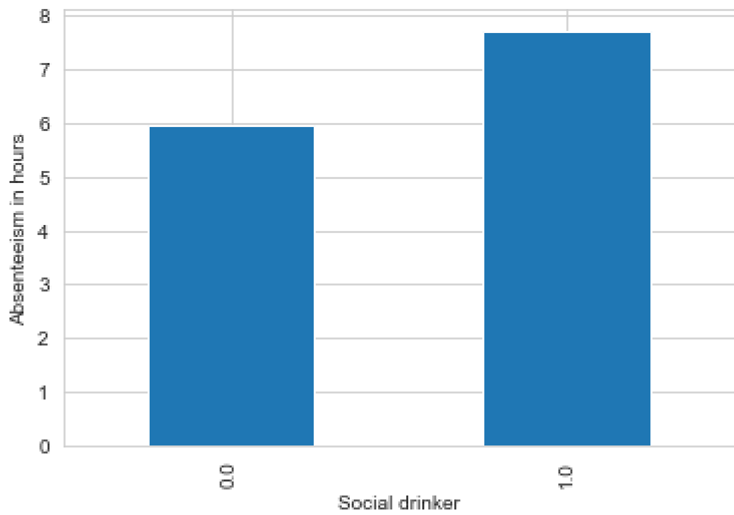
In [20]:
```
# Grouping the data using Pet against our target variable and plotting bar plo
t
emp_data.groupby('Pet').mean()['Absenteeism time in hours'].plot.bar()
plt.ylabel('Absenteeism in hours')
```

Out[20]: Text(0, 0.5, 'Absenteeism in hours')



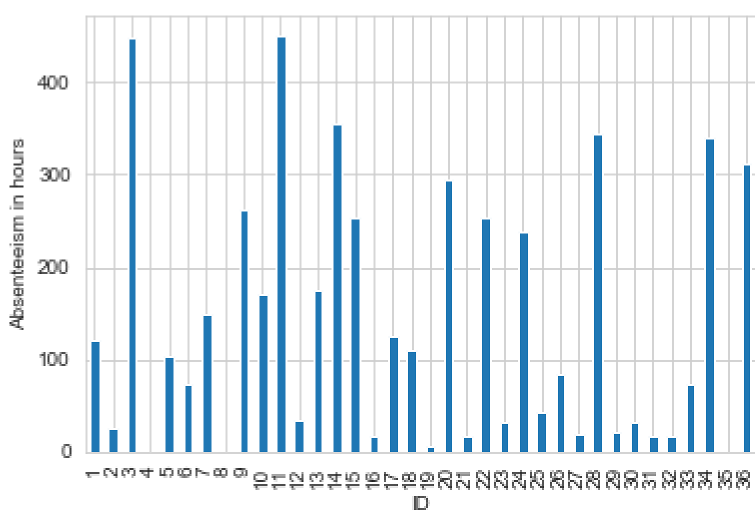
In [21]:
```
# Grouping the data using Social drinker against our target variable and plott
ing bar plot
emp_data.groupby('Social drinker').mean()['Absenteeism time in hours'].plot.ba
r()
plt.ylabel('Absenteeism in hours')
```

Out[21]: Text(0, 0.5, 'Absenteeism in hours')



In [22]:
```
# Grouping the data using ID against our target variable and plotting bar plot
emp_data.groupby('ID').sum()['Absenteeism time in hours'].plot.bar()
plt.ylabel('Absenteeism in hours')
```

Out[22]: Text(0, 0.5, 'Absenteeism in hours')



Missing Value Analysis

To find the missing values and to impute them using the best method - mean, median or KNN Imputation method

In [23]:
```
#To create a dataframe and keep all the missing values in it
missing_values = pd.DataFrame(emp_data.isnull().sum())

#To reset the index
missing_values = missing_values.reset_index()

#To change the names of columns
missing_values = missing_values.rename(columns = {'index' : 'Variables', 0 :
'Missing_Percentage'})

#Calculating the missing values percentage
missing_values['Missing_Percentage'] = (missing_values['Missing_Percentage']/l
en(emp_data)) * 100

#Arrange the dataframe in descending order
missing_values = missing_values.sort_values('Missing_Percentage', ascending =
False).reset_index(drop = True)
```

In [24]:
```
#To verify the dataframe - missing_values
missing_values
```

Out[24]:

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

Imputing the missing values

To impute the missing values into our data, we have three methods - mean, median and KNN imputation. To check the best method, let's create a missing value in any of the continuous variable and proceed with all the three methods. Then pick the best method which is giving the nearest value to the value we removed

In [25]:
```
#Actual value = 28
#Mean Value = 26.68
#Median Value = 25
#KNN value = 28.01
#print(emp_data['Body mass index'].iloc[68])
#Set the value to NaN i.e., creating a misisng value
#emp_data['Body mass index'].iloc[68] = 28
#Impute with the mean method
#emp_data['Body mass index'].iloc[68] = emp_data['Body mass index'].mean()
#Impute with the median method
#emp_data['Body mass index'].iloc[68] = emp_data['Body mass index'].median()

#Imputing with KNN method
emp_data = pd.DataFrame(KNN(k = 3).fit_transform(emp_data), columns = emp_data
.columns)
```

```
Imputing row 1/740 with 0 missing, elapsed time: 0.201
Imputing row 101/740 with 1 missing, elapsed time: 0.204
Imputing row 201/740 with 0 missing, elapsed time: 0.207
Imputing row 301/740 with 0 missing, elapsed time: 0.208
Imputing row 401/740 with 0 missing, elapsed time: 0.210
Imputing row 501/740 with 0 missing, elapsed time: 0.211
Imputing row 601/740 with 0 missing, elapsed time: 0.212
Imputing row 701/740 with 0 missing, elapsed time: 0.213
```

In [26]:
```
#To check for any missing values and ensure no missing values are found
emp_data.isnull().sum()
```

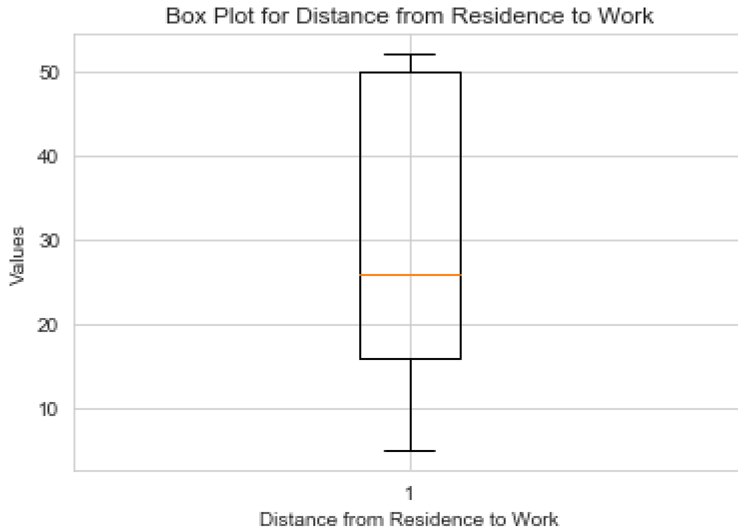
Out[26]:
```
ID                                 0
Reason for absence                 0
Month of absence                   0
Day of the week                    0
Seasons                            0
Transportation expense             0
Distance from Residence to Work    0
Service time                       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
```

Outlier Analysis

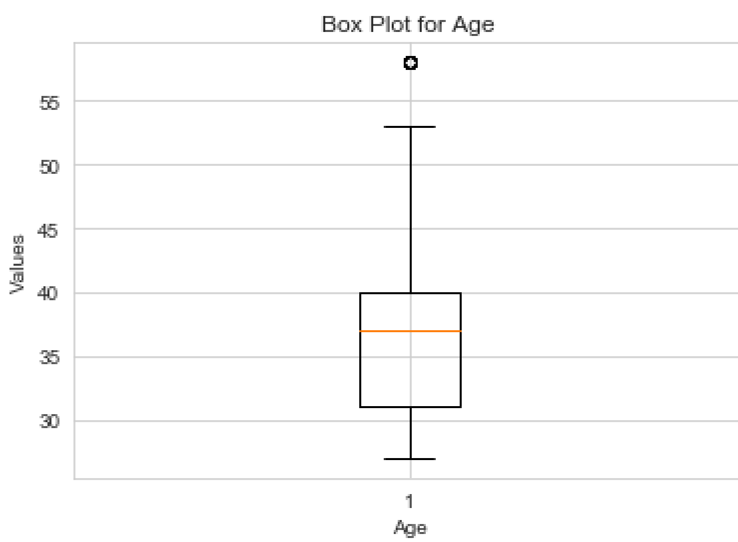
Identify the outliers in continuous variable by plotting box plots, replace them with NA and impute them with KNN method

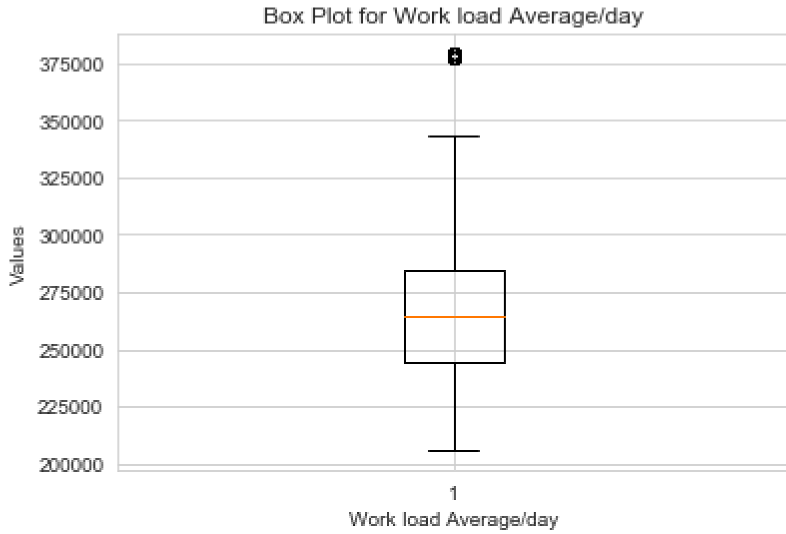
In [27]:
```
#Box plot for Distance from Residence to Work
plt.boxplot(emp_data['Distance from Residence to Work'])
plt.xlabel("Distance from Residence to Work")
plt.ylabel("Values")
plt.title("Box Plot for Distance from Residence to Work")
plt.show()

#Box plot for Service time
plt.boxplot(emp_data['Service time'])
plt.xlabel("Service time")
plt.ylabel("Values")
plt.title("Box Plot for Service time")
plt.show()

#Box plot for Age
plt.boxplot(emp_data['Age'])
plt.xlabel("Age")
plt.ylabel("Values")
plt.title("Box Plot for Age")
plt.show()

#Box plot for Work load Average/day
plt.boxplot(emp_data["Work load Average/day "])
plt.xlabel("Work load Average/day")
plt.ylabel("Values")
plt.title("Box Plot for Work load Average/day")
plt.show()

#Box plot for Transportation expense
plt.boxplot(emp_data['Transportation expense'])
plt.xlabel("Transportation expense")
plt.ylabel("Values")
plt.title("Box Plot for Transportation expense")
plt.show()

#Box plot for Hit target
plt.boxplot(emp_data['Hit target'])
plt.xlabel("Hit target")
plt.ylabel("Values")
plt.title("Box Plot for Hit target")
plt.show()

#Box plot for Weight
plt.boxplot(emp_data['Weight'])
plt.xlabel("Weight")
plt.ylabel("Values")
plt.title("Box Plot for Weight")
plt.show()

#Box plot for Height
plt.boxplot(emp_data['Height'])
plt.xlabel("Height")
plt.ylabel("Values")
plt.title("Box Plot for Height")
plt.show()
```

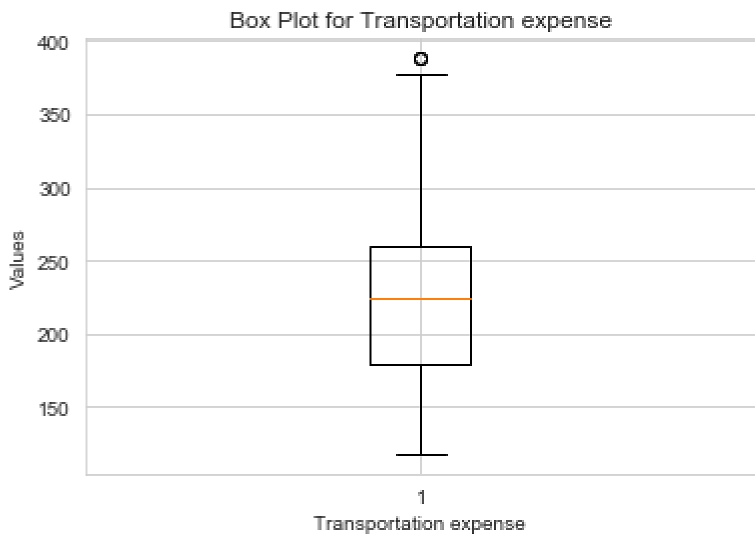
```
#Box plot for Body mass index
plt.boxplot(emp_data['Body mass index'])
plt.xlabel("Body mass index")
plt.ylabel("Values")
plt.title("Box Plot for Body mass index")
plt.show()

#Box plot for Absenteeism time in hours
plt.boxplot(emp_data['Absenteeism time in hours'])
plt.xlabel("Absenteeism time in hours")
plt.ylabel("Values")
plt.title("Box Plot for Absenteeism time in hours")
plt.show()
```

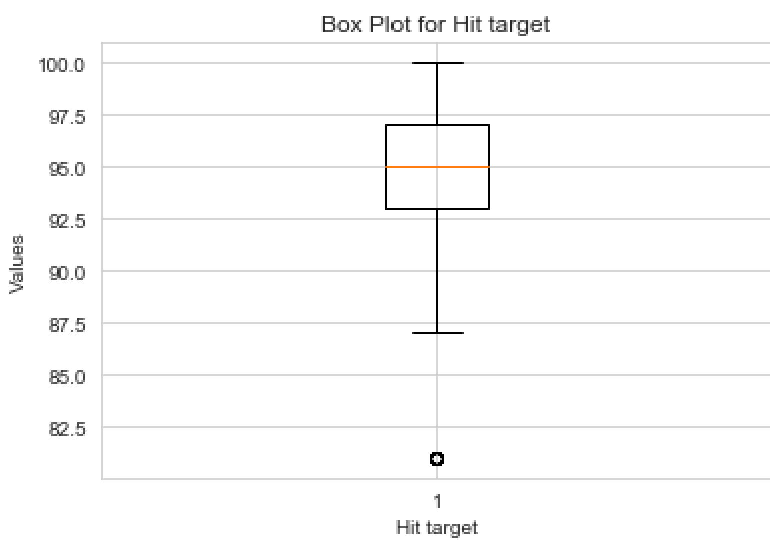

Box Plot for Distance from Residence to Work
Values
50
40
30
20
10
1
Distance from Residence to Work

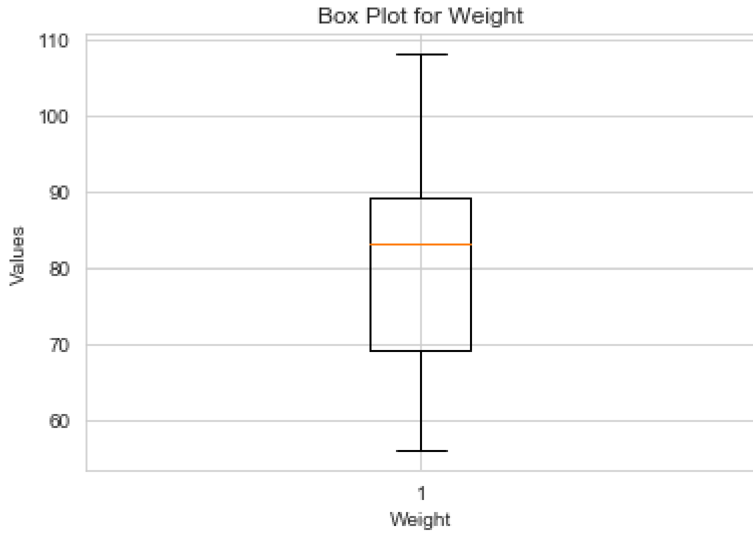

Box Plot for Service time
Values
30
25
20
15
10
5
0
1
Service time

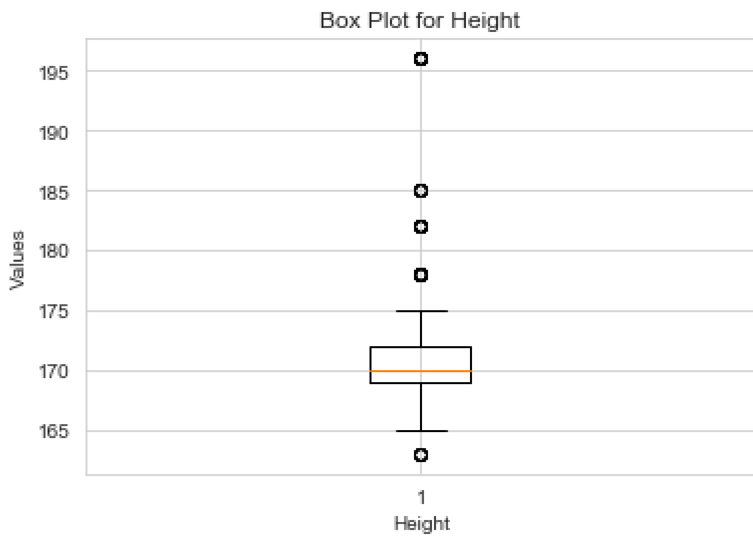

Box Plot for Age
Values
55
50
45
40
35
30
1
Age

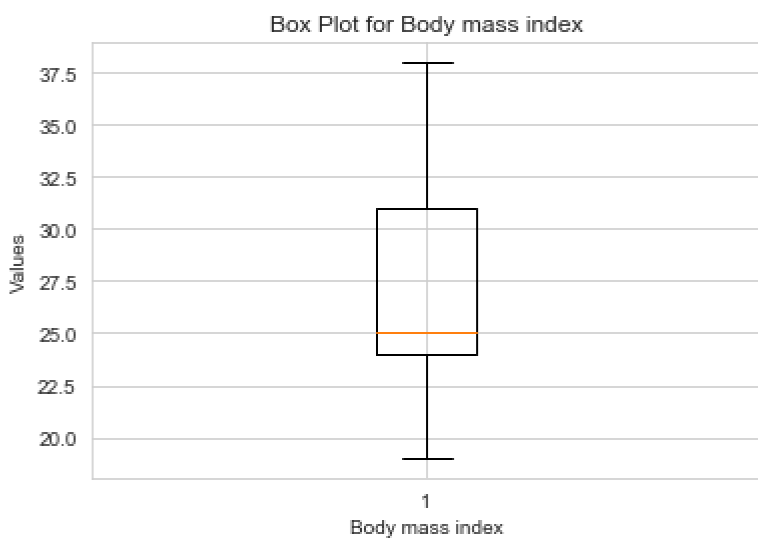

Box Plot for Work load Average/day
Values
375000
350000
325000
300000
275000
250000
225000
200000
1
Work load Average/day

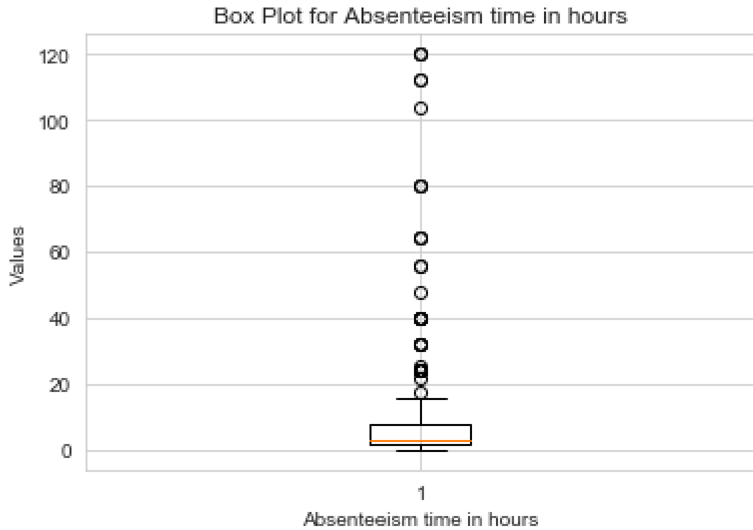

Box Plot for Transportation expense
Values
400
350
300
250
200
150
1
Transportation expense


Box Plot for Hit target
Values
100.0
97.5
95.0
92.5
90.0
87.5
85.0
82.5
1
Hit target


Box Plot for Weight
110
100
90
80
70
60
Values
1
Weight


Box Plot for Height
195
190
185
180
175
170
165
Values
1
Height


Box Plot for Body mass index
37.5
35.0
32.5
30.0
27.5
25.0
22.5
20.0
Values
1
Body mass index



It is clear that variables `Distance from Residence to Work` , `Weight` and `Body mass index` doesn't have any outlier. Hence we should consider all the continuous variables except these three. So let's create a list of these three variables and proceed to detect and remove the values of outliers.

In [28]:
```
#List with variables without outliers
no_outliers = ['Distance from Residence to Work', 'Weight', 'Body mass index']

#Loop through the continuous variables
for i in cont_var:
    if i in no_outliers:
        continue
    q75,q25 = np.percentile(emp_data[i], [75,25]) #To get 75 and 25 percentile values
    iqr = q75 - q25 #Interquartile region

    #Calculating outerfence and innerfence
    outer = q75 + (iqr*1.5)
    inner = q25 - (iqr*1.5)

    # Replacing all the outliers value to NA
    emp_data.loc[emp_data[i]< inner,i] = np.nan
    emp_data.loc[emp_data[i]> outer,i] = np.nan

# Imputing missing values with KNN
emp_data = pd.DataFrame(KNN(k = 3).fit_transform(emp_data), columns = emp_data.columns)
# Checking if there is any missing value
emp_data.isnull().sum().sum()
```

```
Imputing row 1/740 with 0 missing, elapsed time: 0.185
Imputing row 101/740 with 1 missing, elapsed time: 0.187
Imputing row 201/740 with 0 missing, elapsed time: 0.189
Imputing row 301/740 with 0 missing, elapsed time: 0.193
Imputing row 401/740 with 0 missing, elapsed time: 0.194
Imputing row 501/740 with 0 missing, elapsed time: 0.196
Imputing row 601/740 with 0 missing, elapsed time: 0.198
Imputing row 701/740 with 0 missing, elapsed time: 0.200
```

Out[28]: 0

Feature Selection

To check for the multicollinearity for continuous variables by plotting correlation plot and remove the variables with r > 0.8

As our target variable is a continuous variable, we will use a one-way ANOVA. It is used when we have a categorical independent variable (with two or more categories) and a normally distributed interval dependent variable and we wish to test for differences in the means of the dependent variable broken down by the levels of the independent variable.

In [29]:
```
#Correlation analysis for continuous variables
#Let's store all the numeric data into an object
numeric_data = emp_data.loc[:,cont_var]

#Set the measurements of the plot, let's say width = 10 and height = 10

a , k  = plt.subplots(figsize=(10,10))

#Correlation matrix

corr_matrix = numeric_data.corr()

#Plotting a correlation graph
ax = sbn.heatmap(corr_matrix, vmin=-1, vmax=1, center=0, cmap=sbn.diverging_pa
lette(10, 220, n=200),
                 square=True, annot = True)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,horizontalalignment='righ
t')
```

Out[29]:
```
[Text(0.5, 0, 'Distance from Residence to Work'),
 Text(1.5, 0, 'Service time'),
 Text(2.5, 0, 'Age'),
 Text(3.5, 0, 'Work load Average/day '),
 Text(4.5, 0, 'Transportation expense'),
 Text(5.5, 0, 'Hit target'),
 Text(6.5, 0, 'Weight'),
 Text(7.5, 0, 'Height'),
 Text(8.5, 0, 'Body mass index'),
 Text(9.5, 0, 'Absenteeism time in hours')]
```

	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
Distance from Residence to Work	1	0.1	-0.11	-0.083	0.27	0.021	-0.051	-0.13	0.11	0.0035
Service time	0.1	1	0.66	-0.052	-0.39	0.06	0.42	-0.029	0.47	-0.066
Age	-0.11	0.66	1	-0.051	-0.27	-0.016	0.49	-0.045	0.55	-0.079
Work load Average/day	-0.083	-0.052	-0.051	1	-0.026	0.03	-0.09	-0.066	-0.12	0.044
Transportation expense	0.27	-0.39	-0.27	-0.026	1	-0.084	-0.21	-0.033	-0.13	0.2
Hit target	0.021	0.06	-0.016	0.03	-0.084	1	-0.011	0.00036	-0.047	0.02
Weight	-0.051	0.42	0.49	-0.09	-0.21	-0.011	1	0.11	0.9	-0.015
Height	-0.13	-0.029	-0.045	-0.066	-0.033	0.00036	0.11	1	-0.079	0.049
Body mass index	0.11	0.47	0.55	-0.12	-0.13	-0.047	0.9	-0.079	1	-0.039
Absenteeism time in hours	0.0035	-0.066	-0.079	0.044	0.2	0.02	-0.015	0.049	-0.039	1

In [30]:
```
#ANOVA test between all the categorical variables to the target variable
for i in cat_var:
    f, p = stats.f_oneway(emp_data[i],emp_data["Absenteeism time in hours"])
    print("P values for variable " + str(i) + " is " + str(p) )
```

```
P values for variable ID is 1.6079132646808366e-172
P values for variable Reason for absence is 3.196042855164201e-274
P values for variable Month of absence is 1.6977845335995727e-27
P values for variable Day of the week is 0.0005184236142440631
P values for variable Seasons is 3.830937816907899e-42
P values for variable Disciplinary failure is 1.1530491381088065e-193
P values for variable Education is 1.6102259682550297e-110
P values for variable Social drinker is 1.3598436285429815e-157
P values for variable Social smoker is 4.201173983633963e-192
P values for variable Pet is 1.0050750974933867e-132
P values for variable Son is 7.220350139758658e-121
```

In [31]:
```
#Dropping the variables with high correlation and much redundation
drop_var = ['Weight']
emp_data = emp_data.drop(drop_var, axis = 1)
emp_data.shape
```

Out[31]: (740, 20)

In [32]:
```
#Updating the continuous variables
del(cont_var[6])
print(cont_var)
```

```
['Distance from Residence to Work', 'Service time', 'Age', 'Work load Averag
e/day ', 'Transportation expense', 'Hit target', 'Height', 'Body mass index',
'Absenteeism time in hours']
```

In [33]:
```
#To have a copy of cleaned data
data_absent = emp_data.copy()
data_absent.shape
```

Out[33]: (740, 20)

Feature Scaling

To scale all the numeric data in between the values of 0 and 1 As seen from the above plots, we can conclude that no continuous variable is having a normal or uniform distribution. So let's proceed with the normalization technique.

In [34]:
```
for i in cont_var:
    if i == 'Absenteeism time in hours' :
        continue
    emp_data[i] = (emp_data[i] - emp_data[i].min()) / (emp_data[i].max() - emp
_data[i].min())#Normalization formula
```

In [35]:
```
#To verify if all the numeric variables have values between 0 and 1
emp_data.head()
```

Out[35]:

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	W Ave
0	11.0	26.0	7.0	3.0	1.0	0.657692	0.659574	0.521739	0.230769	
1	36.0	0.0	7.0	3.0	1.0	0.000000	0.170213	0.739130	0.884615	
2	3.0	23.0	7.0	4.0	1.0	0.234615	0.978723	0.739130	0.423077	
3	7.0	7.0	7.0	5.0	1.0	0.619231	0.000000	0.565217	0.461538	
4	11.0	23.0	7.0	5.0	1.0	0.657692	0.659574	0.521739	0.230769	

Let's create dummy variables for categorical variables.

In [36]:
```
#Get dummy variables for categorical variables
emp_data = pd.get_dummies(emp_data, columns = cat_var)
emp_data.shape
```

Out[36]: (740, 134)

In [37]:
```
emp_data.head()
```

Out[37]:

	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Hit target	Height	Body mass index	Al
0	0.657692	0.659574	0.521739	0.230769	0.244925	0.769231	0.700000	0.578947	
1	0.000000	0.170213	0.739130	0.884615	0.244925	0.769231	0.500001	0.631579	
2	0.234615	0.978723	0.739130	0.423077	0.244925	0.769231	0.500000	0.631579	
3	0.619231	0.000000	0.565217	0.461538	0.244925	0.769231	0.300000	0.263158	
4	0.657692	0.659574	0.521739	0.230769	0.244925	0.769231	0.700000	0.578947	

5 rows × 134 columns

Model development

We've performed all the **Preprocessing techniques** for our data. Our next step is to divide the data into train and test, build a model upon the train data and evaluate on the test data

In [38]:
```
#Splitting into train and test data
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(emp_data.iloc[:,emp_data.colu
mns != 'Absenteeism time in hours'],
                                                  emp_data.iloc[:, 8], test_siz
e = 0.20, random_state = 1)
```

Linear Regression

Root mean squared error :- 30297628104509.066 , R_Squared value :- -7.824448752340836e+25

In [39]:
```
#Building the model by using linear regression
#Importing the necessary libraries for Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

#Build a model on our training dataset
lr_model = LinearRegression().fit(X_train,y_train)

#Predict for the test cases
lr_predictions = lr_model.predict(X_test)

#Let's create a dataframe for both actual and predicted values
df_lrmodel = pd.DataFrame({"Actual"  : y_test, "Predicted" : lr_predictions})
print(df_lrmodel.head())

#Function to find RMSE

def RMSE(x,y):
    rmse = np.sqrt(mean_squared_error(x,y))
    return rmse

#Calculate RMSE and R-Squared value
print("Root mean squared error :- " + str(RMSE(y_test,lr_predictions)))
print("R_Squared value :- " + str(r2_score(y_test,lr_predictions)))
```

```
     Actual  Predicted
681     8.0    4.81250
257     2.0    4.43750
527     8.0    7.93750
637     8.0    4.65625
429     4.0    3.50000
Root mean squared error :- 30297628104509.066
R_Squared value :- -7.824448752340836e+25
```

Random Forest

Root Mean Squared Error: 2.8934271438589225, R_square Score: 0.2863889073811593

In [40]:
```
#Import library for RandomForest
from sklearn.ensemble import RandomForestRegressor
#Build random forest using RandomForestRegressor
ranfor_model = RandomForestRegressor(n_estimators = 300, random_state = 1).fit
(X_train,y_train)

#Perdict for test cases
rf_predictions = ranfor_model.predict(X_test)

#Create data frame for actual and predicted values
df_rf = pd.DataFrame({'Actual': y_test, 'Predicted': rf_predictions})
print(df_rf.head())

#Calculate RMSE and R-squared value
print("Root Mean Squared Error: "+str(RMSE(y_test, rf_predictions)))
print("R_square Score: "+str(r2_score(y_test, rf_predictions)))
```

```
     Actual  Predicted
681     8.0   4.997533
257     2.0   3.664300
527     8.0   5.557778
637     8.0   3.558774
429     4.0   3.886556
Root Mean Squared Error: 2.8934271438589225
R_square Score: 0.2863889073811593
```

Decision Tree

RMSE: 3.3385763215701907 , R_Square score: 0.04992233776575228

In [41]:
```
#Importing necessary libraries for Decision tree
from sklearn.tree import DecisionTreeRegressor

#Build Decision tree model on the train data
dt_model = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)

#Predict for the test cases
dt_predict = dt_model.predict(X_test)

#Create a dataframe for actual and predicted values
df_dtmodel = pd.DataFrame({"Actual" : y_test, "Predicted" : dt_predict})
print(df_dtmodel.head())

#Calculate RMSE and R_squared values
print("RMSE: " + str(RMSE(y_test,dt_predict)))
print("R_Square score: " + str(r2_score(y_test,dt_predict)))
```

```
     Actual  Predicted
681     8.0   5.152237
257     2.0   5.152237
527     8.0   2.899160
637     8.0   5.152237
429     4.0   5.152237
RMSE: 3.3385763215701907
R_Square score: 0.04992233776575228
```

Gradient Boosting

RMSE: 3.3385763215701907 , **R_Square score: 0.04992233776575228**

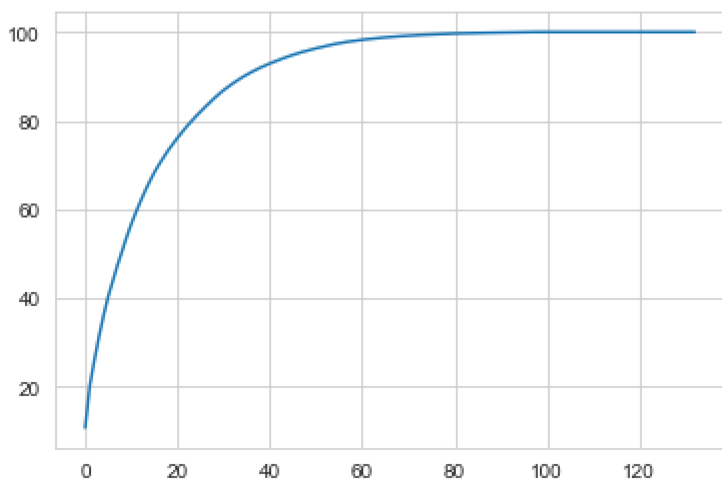
In [42]:
```
#Import necessary libraries for this ML algorithm
from sklearn.ensemble import GradientBoostingRegressor

#Build GB model on the train data
gb_model = GradientBoostingRegressor().fit(X_train,y_train)

#Predict the test cases
gb_predict = gb_model.predict(X_test)

#Create a dataframe for actual and predicted values
df_gbmodel = pd.DataFrame({"Actual" : y_test, "Predicted" : dt_predict})
print(df_gbmodel.head())

#Calculate RMSE and R_squared values
print("RMSE: " + str(RMSE(y_test,dt_predict)))
print("R_Square score: " + str(r2_score(y_test,dt_predict)))
```

```
     Actual  Predicted
681     8.0   5.152237
257     2.0   5.152237
527     8.0   2.899160
637     8.0   5.152237
429     4.0   5.152237
RMSE: 3.3385763215701907
R_Square score: 0.04992233776575228
```

Dimension Reduction using Pricipal Component Analysis

Principal Component Analysis (PCA) is a dimension-reduction tool that can be used to reduce a large set of variables to a small set that still contains most of the information in the large set.

In [43]:
```
#Get the target variable
target_var = emp_data['Absenteeism time in hours']

#Get the shape of our cleaned dataset
emp_data.shape #740 134

#Importing the library for PCA
from sklearn.decomposition import PCA

#Dropping the target variable
emp_data.drop(['Absenteeism time in hours'], inplace = True, axis =1)

#To check the shape of the data after dropping the target variable
emp_data.shape# 740 133

#Converting our data to numpy array
numpy_data = emp_data.values

#Our data without target variable has 133 variables, so number of components = 
133

pca = PCA(n_components = 133)
pca.fit(numpy_data)

#To check the variance that each PC explains
var = pca.explained_variance_ratio_

#Cumulative variance
var_cum = np.cumsum(np.round(var, decimals = 4) * 100)

plt.plot(var_cum)
plt.show()
```



From the above graph, it is clear that approximately after 50 components, there is no variance even if all the rest of the components are considered. So let's select these 50 components as it explains almost 95 percent data variance.

In [47]:
```
#Selecting the 50 components
pca = PCA(n_components = 50)

#To fit the selected components to the data
pca.fit(numpy_data)

#Splitting into train and test data using train_test_split
X_train,X_test,y_train,y_test = train_test_split(numpy_data,target_var,test_si
ze = 0.2)
```

Now by using the above data let's develop the model by using various machine learning algorithms.

Linear Regression

Root mean squared error :- 1700235342812.101 , R_Squared value :- -2.4840239176568323e+23

In [48]:
```
#Building the model by using linear regression
#Importing the necessary libraries for Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

#Build a model on our training dataset
lr_model = LinearRegression().fit(X_train,y_train)

#Predict for the test cases
lr_predictions = lr_model.predict(X_test)

#Let's create a dataframe for both actual and predicted values
df_lrmodel = pd.DataFrame({"Actual"  : y_test, "Predicted" : lr_predictions})
print(df_lrmodel.head())

#Function to find RMSE

def RMSE(x,y):
    rmse = np.sqrt(mean_squared_error(x,y))
    return rmse

#Calculate RMSE and R-Squared value
print("Root mean squared error :- " + str(RMSE(y_test,lr_predictions)))
print("R_Squared value :- " + str(r2_score(y_test,lr_predictions)))
```

```
     Actual  Predicted
92      3.0   3.881348
633     2.0   3.275879
207     8.0   6.994629
340     2.0   2.807129
496     1.0   4.357910
Root mean squared error :- 1700235342812.101
R_Squared value :- -2.4840239176568323e+23
```

RandomForest

Root Mean Squared Error: 2.6513491672836627 , R_square Score: 0.3959518687970416

In [49]:
```
#Import library for RandomForest
from sklearn.ensemble import RandomForestRegressor
#Build random forest using RandomForestRegressor
ranfor_model = RandomForestRegressor(n_estimators = 300, random_state = 1).fit
(X_train,y_train)

#Perdict for test cases
rf_predictions = ranfor_model.predict(X_test)

#Create data frame for actual and predicted values
df_rf = pd.DataFrame({'Actual': y_test, 'Predicted': rf_predictions})
print(df_rf.head())

#Calculate RMSE and R-squared value
print("Root Mean Squared Error: "+str(RMSE(y_test, rf_predictions)))
print("R_square Score: "+str(r2_score(y_test, rf_predictions)))
```

```
     Actual  Predicted
92      3.0   3.497758
633     2.0   2.620000
207     8.0   6.285799
340     2.0   1.575972
496     1.0   1.560000
Root Mean Squared Error: 2.6513491672836627
R_square Score: 0.3959518687970416
```

Decision Tree

RMSE: 3.1792364568869935 , R_Square score: 0.13147294034144996

In [50]:
```
#Importing necessary libraries for Decision tree
from sklearn.tree import DecisionTreeRegressor

#Build Decision tree model on the train data
dt_model = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)

#Predict for the test cases
dt_predict = dt_model.predict(X_test)

#Create a dataframe for actual and predicted values
df_dtmodel = pd.DataFrame({"Actual" : y_test, "Predicted" : dt_predict})
print(df_dtmodel.head())

#Calculate RMSE and R_squared values
print("RMSE: " + str(RMSE(y_test,dt_predict)))
print("R_Square score: " + str(r2_score(y_test,dt_predict)))
```

```
     Actual  Predicted
92      3.0   3.773049
633     2.0   3.773049
207     8.0   3.773049
340     2.0   3.773049
496     1.0   5.611381
RMSE: 3.1792364568869935
R_Square score: 0.13147294034144996
```

Gradient Boosting

RMSE: 3.1792364568869935 , **R_Square score: 0.13147294034144996**

In [51]:
```
#Import necessary libraries for this ML algorithm
from sklearn.ensemble import GradientBoostingRegressor

#Build GB model on the train data
gb_model = GradientBoostingRegressor().fit(X_train,y_train)

#Predict the test cases
gb_predict = gb_model.predict(X_test)

#Create a dataframe for actual and predicted values
df_gbmodel = pd.DataFrame({"Actual" : y_test, "Predicted" : dt_predict})
print(df_gbmodel.head())

#Calculate RMSE and R_squared values
print("RMSE: " + str(RMSE(y_test,dt_predict)))
print("R_Square score: " + str(r2_score(y_test,dt_predict)))
```

```
     Actual  Predicted
92      3.0   3.773049
633     2.0   3.773049
207     8.0   3.773049
340     2.0   3.773049
496     1.0   5.611381
RMSE: 3.1792364568869935
R_Square score: 0.13147294034144996
```

In [ ]: