Impact Sure Assignment

Case Study - Absent at work

Arbaaz Mehboob Sayyed.

ROll No- 21.

Jai Hind College , Mumbai.

In [86]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# Installing Essentials libraries
```

In [87]:
```
data=pd.read_csv("Absenteeism_at_work",delimiter=';')

#importing the csv dataset with ; as seprator
```

In [88]:
```
data.head()

#displaying top head of the dataset.
```

Out[88]:

	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	...	Dis
0	11	26	7	3	1	289	36	13	33	239.554	...	
1	36	0	7	3	1	118	13	18	50	239.554	...	
2	3	23	7	4	1	179	51	18	38	239.554	...	
3	7	7	7	5	1	279	5	14	39	239.554	...	
4	11	23	7	5	1	289	36	13	33	239.554	...	

5 rows × 21 columns

In [89]:
```
data.shape

#Shape i.e No.of rows and columns in the data Rows=740 and 21=Cols
```

Out[89]:
```
(740, 21)
```

In [90]:
```python
data.columns

#columns or features name
```

Out[90]:
```
Index(['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 ', 'Hit target',
       'Disciplinary failure', 'Education', 'Son', 'Social drinker',
       'Social smoker', 'Pet', 'Weight', 'Height', 'Body mass index',
       'Absenteeism time in hours'],
      dtype='object')
```

In [91]:
```python
data.info()

#depth information about data and features in it.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 740 entries, 0 to 739
Data columns (total 21 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   ID                               740 non-null    int64  
 1   Reason for absence               740 non-null    int64  
 2   Month of absence                 740 non-null    int64  
 3   Day of the week                  740 non-null    int64  
 4   Seasons                          740 non-null    int64  
 5   Transportation expense           740 non-null    int64  
 6   Distance from Residence to Work  740 non-null    int64  
 7   Service time                     740 non-null    int64  
 8   Age                              740 non-null    int64  
 9   Work load Average/day            740 non-null    float64
 10  Hit target                       740 non-null    int64  
 11  Disciplinary failure             740 non-null    int64  
 12  Education                        740 non-null    int64  
 13  Son                              740 non-null    int64  
 14  Social drinker                   740 non-null    int64  
 15  Social smoker                    740 non-null    int64  
 16  Pet                              740 non-null    int64  
 17  Weight                           740 non-null    int64  
 18  Height                           740 non-null    int64  
 19  Body mass index                  740 non-null    int64  
 20  Absenteeism time in hours        740 non-null    int64  
dtypes: float64(1), int64(20)
memory usage: 121.5 KB
```

In [92]:
```python
data.describe(include="all")

#to check the summary of the data, describe() methods return mean count std quartiles and
```

Out[92]:

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time
count	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000
mean	18.017568	19.216216	6.324324	3.914865	2.544595	221.329730	29.631081	12.554054
std	11.021247	8.433406	3.436287	1.421675	1.111831	66.952223	14.836788	4.384873
min	1.000000	0.000000	0.000000	2.000000	1.000000	118.000000	5.000000	1.000000

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time
25%	9.000000	13.000000	3.000000	3.000000	2.000000	179.000000	16.000000	9.000000
50%	18.000000	23.000000	6.000000	4.000000	3.000000	225.000000	26.000000	13.000000
75%	28.000000	26.000000	9.000000	5.000000	4.000000	260.000000	50.000000	16.000000
max	36.000000	28.000000	12.000000	6.000000	4.000000	388.000000	52.000000	29.000000

8 rows × 21 columns

In [93]:

```
data.nunique()

# Number of unique items in particular feature.
```

Out[93]:

```
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 [94]:

```
missing_values=data.isnull().sum()

percentage=data.isnull().sum()/data.shape[0]*100

value={
    "missing_values  ":missing_values,
    "percent_of_missing ":percentage
}

frame=pd.DataFrame(value)
frame

# Check whether there is some missing values in the data by taking out the percentage . i
```

Out[94]:

	missing_values	percent_of_missing

	missing_values	percent_of_missing
ID	0	0.0
Reason for absence	0	0.0
Month of absence	0	0.0
Day of the week	0	0.0
Seasons	0	0.0
Transportation expense	0	0.0
Distance from Residence to Work	0	0.0
Service time	0	0.0
Age	0	0.0
Work load Average/day	0	0.0
Hit target	0	0.0
Disciplinary failure	0	0.0
Education	0	0.0
Son	0	0.0
Social drinker	0	0.0
Social smoker	0	0.0
Pet	0	0.0
Weight	0	0.0
Height	0	0.0
Body mass index	0	0.0
Absenteeism time in hours	0	0.0

In [95]:

```
data.columns
```

Out[95]:

```
Index(['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 ', 'Hit target',
       'Disciplinary failure', 'Education', 'Son', 'Social drinker',
       'Social smoker', 'Pet', 'Weight', 'Height', 'Body mass index',
       'Absenteeism time in hours'],
      dtype='object')
```

In [96]:

```
data.rename(columns={'Work load Average/day ':'Workload_Avg_Day',
                     'Reason for absence':'Reason_of_Absence',
                     'Month of absence':'Month_of_Absence',
                     'Day of the week':'Day',
                     'Seasons':'Season',
                     'Transportation expense':'Transportation_expense',
                     'Distance from Residence to Work':'Distance_home',
                     'Service time':'Service_time',
                     'Hit target':'Hit_target',
                     'Disciplinary failure':'Disciplinary_failure',
                     'Social drinker':'Social_drinker',
```

```
                    'Social smoker':'Social_smoker',
                    'Body mass index':'BMI',
                    'Absenteeism time in hours':'Absent'

                    },
            inplace=True)

# Renaming the columns for easy analysis.
```

In [97]:

```
data
```

Out[97]:

	ID	Reason_of_Absence	Month_of_Absence	Day	Season	Transportation_expense	Distance_home	S
0	11	26	7	3	1	289	36	
1	36	0	7	3	1	118	13	
2	3	23	7	4	1	179	51	
3	7	7	7	5	1	279	5	
4	11	23	7	5	1	289	36	
...	...	...	...	...	...	...	...	
735	11	14	7	3	1	289	36	
736	1	11	7	3	1	235	11	
737	4	0	0	3	1	118	14	
738	8	0	0	4	2	231	35	
739	35	0	0	6	3	179	45	

740 rows × 21 columns

In [99]:

```
data.drop(columns='ID',inplace=True)

# removing the ID feature.
```

In [100]:

```
data.columns
```

Out[100]:

```
Index(['Reason_of_Absence', 'Month_of_Absence', 'Day', 'Season',
       'Transportation_expense', 'Distance_home', 'Service_time', 'Age',
       'Workload_Avg_Day', 'Hit_target', 'Disciplinary_failure', 'Education',
       'Son', 'Social_drinker', 'Social_smoker', 'Pet', 'Weight', 'Height',
       'BMI', 'Absent'],
      dtype='object')
```

In [101]:

```
print("Unique Values of reason of Response :",data['Reason_of_Absence'].unique())
print("Month of Ansence in year :",data['Month_of_Absence'].unique())
print("Days in week :",data['Day'].unique())
print("Expense for commuting:",data['Transportation_expense'].unique())
print("Season :",data['Season'].unique())
print("Distance from Home :",data['Distance_home'].unique())
print("Service_time :",data['Service_time'].unique())
```

```
print("Age group people :",data['Age'].unique())
print("Workload :",data['Workload_Avg_Day'].unique())
print("Hit target :",data['Hit_target'].unique())
print("Disciplinary_failure :",data['Disciplinary_failure'].unique())
print("Education :",data['Education'].unique())
print("Son :",data['Son'].unique())
print("social smoker :",data['Social_smoker'].unique())

print("social drinker :",data['Social_drinker'].unique())

print("Pet :",data['Pet'].unique())
print("Weight :",data['Weight'].unique())
print("Height :",data['Height'].unique())
print("BMI :",data['BMI'].unique())
print("Absent :",data['Absent'].unique())



# Exploring the Unique values amongst all the features available.
```

```
Unique Values of reason of Response : [26  0 23  7 22 19  1 11 14 21 10 13 28 18 25 24  6
27 17  8 12  5  9 15
  4  3  2 16]
Month of Ansence in year : [ 7  8  9 10 11 12  1  2  3  4  5  6  0]
Days in week : [3 4 5 6 2]
Expense for commuting: [289 118 179 279 361 260 155 235 246 189 248 330 157 291 184 225 3
69 388
 378 228 300 268 231 233]
Season : [1 4 2 3]
Distance from Home : [36 13 51  5 52 50 12 11 25 29 16 27 42 10 20 31 26 17 22 15 49 48 1
4 35
 45]
Service_time : [13 18 14  3 11 16  4  6 12  7 10  9 17 29  8  1 15 24]
Age group people : [33 50 38 39 28 36 34 37 41 47 29 48 32 27 43 40 31 30 49 58 46 53]
Workload : [239.554 205.917 241.476 253.465 306.345 261.306 308.593 302.585 343.253
 326.452 378.884 377.55  275.312 265.615 294.217 265.017 284.031 236.629
 330.061 251.818 244.387 239.409 246.074 253.957 230.29  249.797 261.756
 284.853 268.519 280.549 313.532 264.249 222.196 246.288 237.656 275.089
 264.604 271.219]
Hit target : [ 97  92  93  95  99  96  94  98  81  88 100  87  91]
Disciplinary_failure : [0 1]
Education : [1 3 2 4]
Son : [2 1 0 4 3]
social smoker : [0 1]
social drinker : [1 0]
Pet : [1 0 4 2 5 8]
Weight : [ 90  98  89  68  80  65  95  88  67  69  86  84  75  58  83 106  73  70
  56  63  76 108  77  79 100  94]
Height : [172 178 170 168 196 167 165 182 185 163 169 171 174 175]
BMI : [30 31 24 27 23 25 29 32 22 33 21 28 38 19 36 35 34]
Absent : [  4   0   2   8  40   1   7   3  32   5  16  24  64  56  80 120 112 104
  48]
```

Exploratory Data Analysis

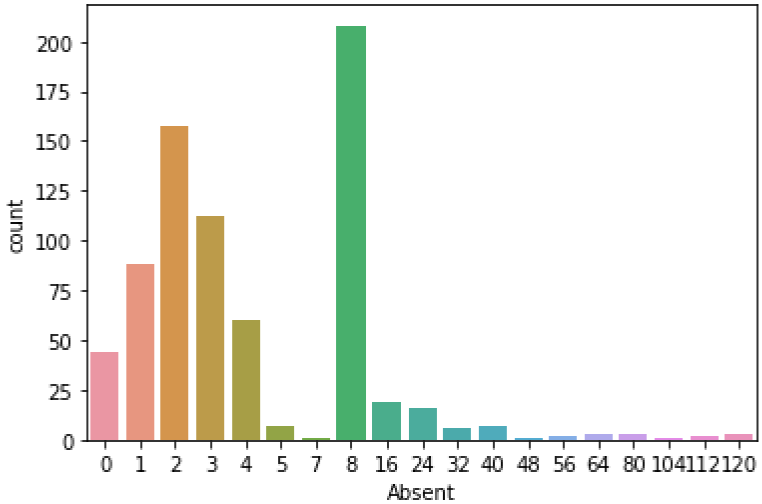
In [28]:

```
sns.countplot(x=data['Absent'],data=data)

# Checking number of hours agent absent at work.
```

```
<AxesSubplot:xlabel='Absent', ylabel='count'>
```

Out[28]:



Mostly the delivery agents are absent for an hour or two or three but the major population of agents absent for 8 hours.

In [104]:

```
feat=['Social_drinker','Social_smoker','Disciplinary_failure',
      'Month_of_Absence','Day','Season','Education','Son','Pet']
list(enumerate(feat))

#Countplot for categorical features.
```

Out[104]:

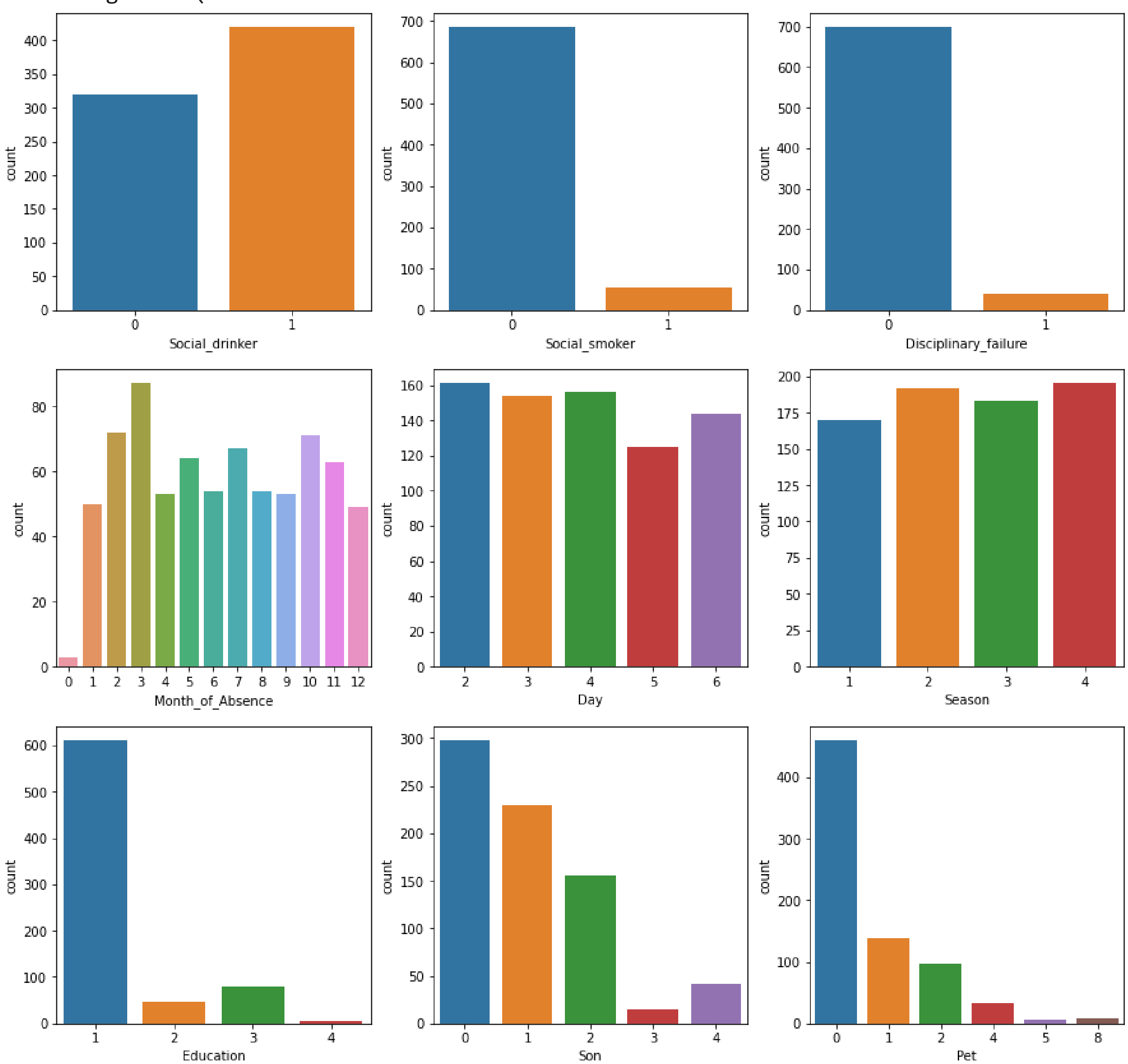
```
[(0, 'Social_drinker'),
 (1, 'Social_smoker'),
 (2, 'Disciplinary_failure'),
 (3, 'Month_of_Absence'),
 (4, 'Day'),
 (5, 'Season'),
 (6, 'Education'),
 (7, 'Son'),
 (8, 'Pet')]
```

In [105]:

```
plt.figure(figsize=(15,30))
for i in enumerate(feat):
    plt.subplot(6,3,i[0]+1)
    sns.countplot(i[1],data=data)
```

```
C:\Users\asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
C:\Users\asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
C:\Users\asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
C:\Users\asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
```

```
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
C:\Users\asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
C:\Users\asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
C:\Users\asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
C:\Users\asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```



1.Social Drinker- (Yes=1,No=0) There are more number of social drinkers.

2.Social Smoker- (Yes=1,No=0) the non smoker category agents are comparatively larger in quantity then smokers.

3.Disciplinary Failure - (Yes=1,No=0) misconduct is less than although there is some failure.

4.Month of Absence - (1 to 12 serial order months Jan to Dec) Month of march have more absent agents at work.

5.Day of week -(2- Mon,3-Tue,4-wed,5-thu,6-fri) there is all around work and utmost data for days in week.

1. Season - (four season 1 to 4).

7.Education -(Highschool, Graduate,PostGraduate, Masters and pHd) most of the agents are from high school.

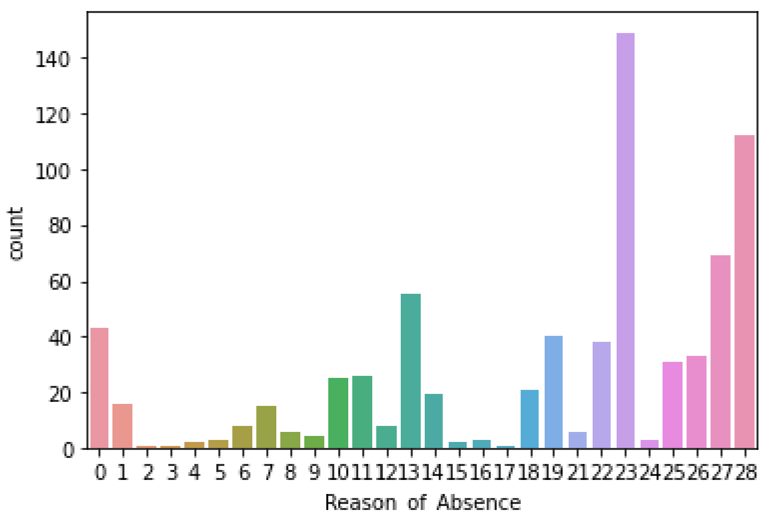
8.Son - (1 to 4 sons) Most of the agents have no offsprings or be an single.

1. Pet - Most of the agents dont have pets.

In [31]:

```
sns.countplot(x=data['Reason_of_Absence'],data=data)
```

Out[31]: <AxesSubplot:xlabel='Reason_of_Absence', ylabel='count'>



Most Number of the agents were absent due to the reason of MEDICAL Consultation Followed by Dental Consultation , Physiotherpy and disease of muscoskeletal system and connective tissue problems

In [111]:

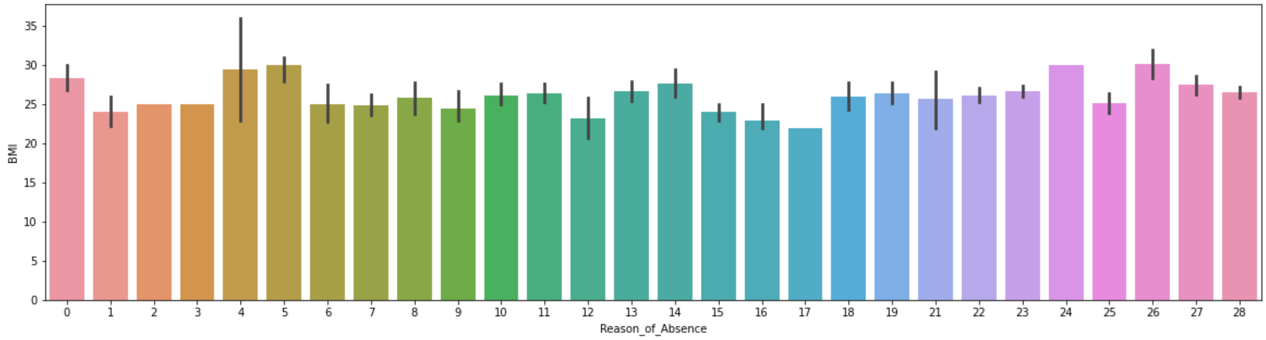
```
data.columns
```

Out[111]:

```
Index(['Reason_of_Absence', 'Month_of_Absence', 'Day', 'Season',
       'Transportation_expense', 'Distance_home', 'Service_time', 'Age',
       'Workload_Avg_Day', 'Hit_target', 'Disciplinary_failure', 'Education',
       'Son', 'Social_drinker', 'Social_smoker', 'Pet', 'Weight', 'Height',
       'BMI', 'Absent'],
      dtype='object')
```

In [112]:
```
plt.figure(figsize=(20,5))
sns.barplot(x=data['Reason_of_Absence'],y=data['BMI'],data=data)
```

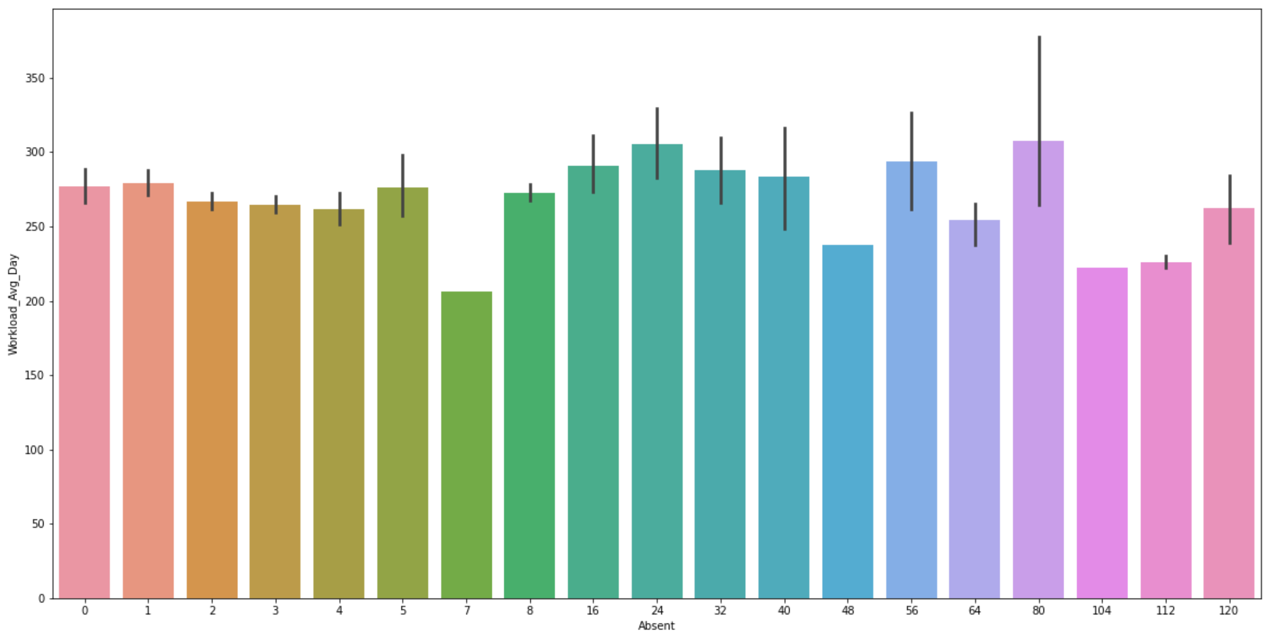
Out[112]: <AxesSubplot:xlabel='Reason_of_Absence', ylabel='BMI'>



Bivariate analysis of visualizing the Reason of Absence to the BMI data. Ideal BMI is 18.5 to 25. hence most of the reason for absence is for higher BMI agents i.e obese category.

In [37]:
```
plt.figure(figsize=(20,10))
sns.barplot(x=data['Absent'],y=data['Workload_Avg_Day'],data=data)
```

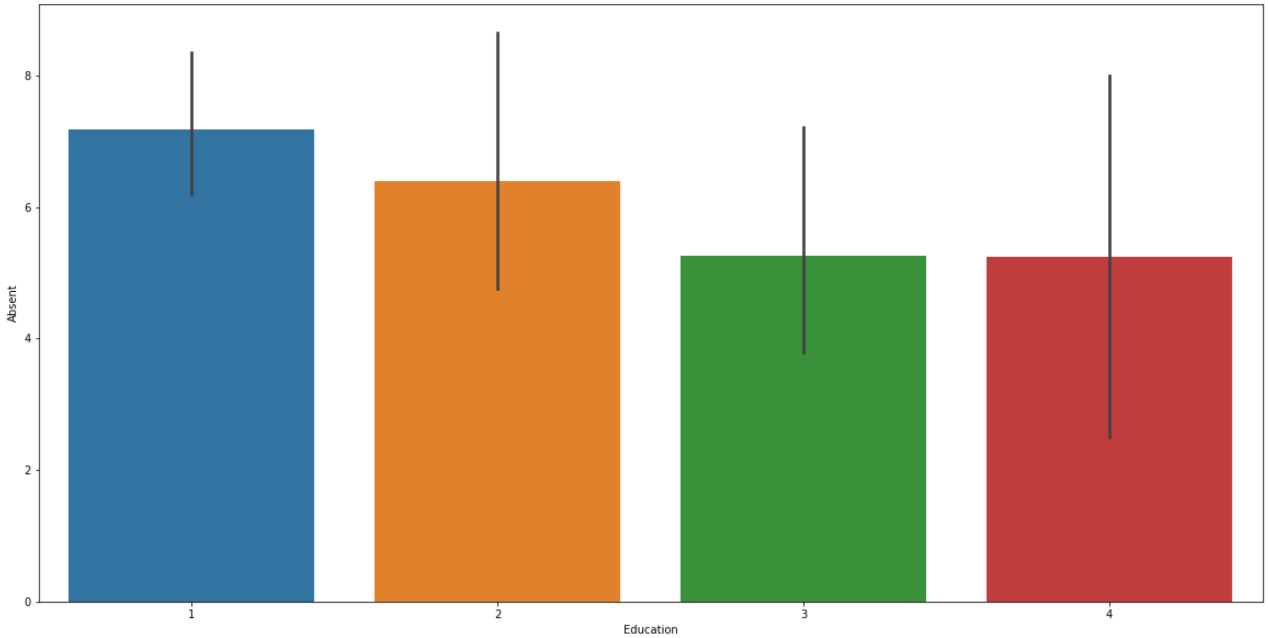
Out[37]: <AxesSubplot:xlabel='Absent', ylabel='Workload_Avg_Day'>



Visualizing the Workload Index to Absent at work data.

In [38]:
```
plt.figure(figsize=(20,10))
sns.barplot(x=data['Education'],y=data['Absent'],data=data)
```

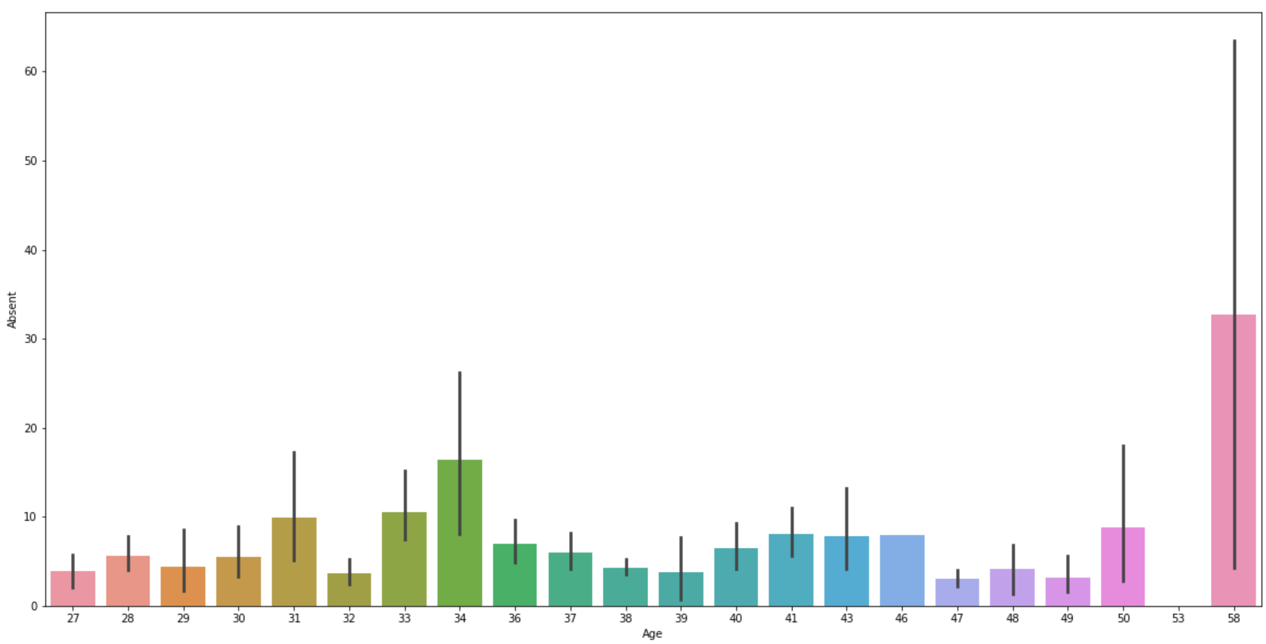
Out[38]: <AxesSubplot:xlabel='Education', ylabel='Absent'>



In [39]:

```
plt.figure(figsize=(20,10))
sns.barplot(x=data['Age'],y=data['Absent'],data=data)
```

Out[39]: <AxesSubplot:xlabel='Age', ylabel='Absent'>

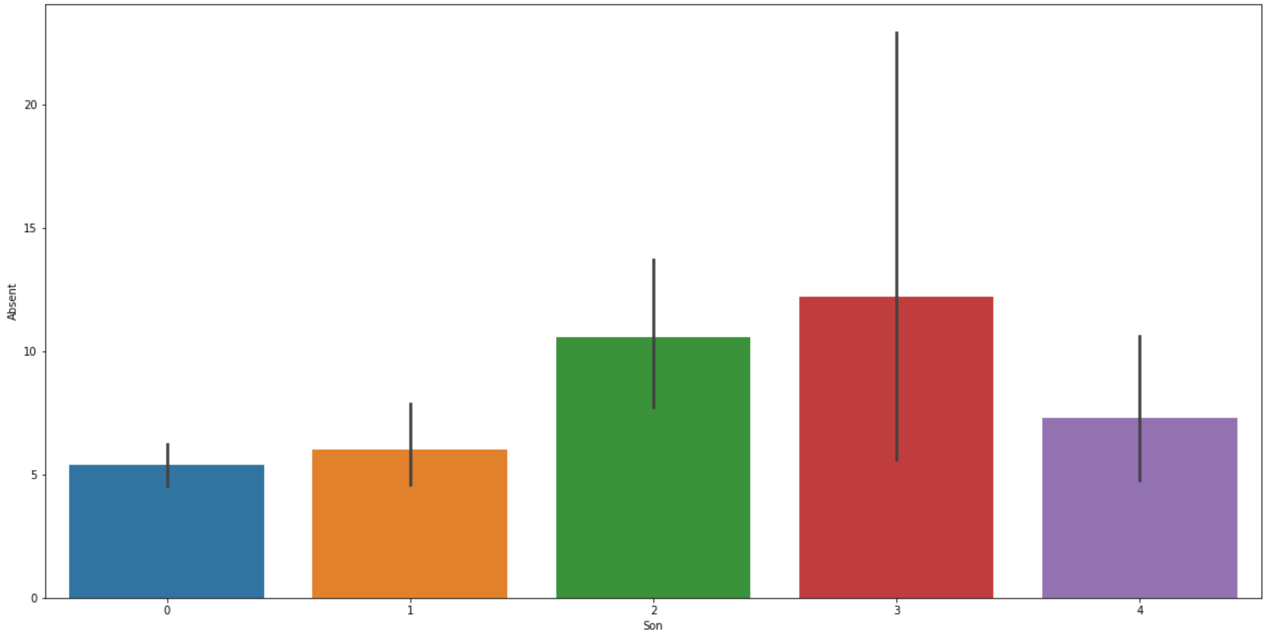


Agents from age group 31-35 tends to be more absent at work and as well from 50+ category.

In [40]:

```
plt.figure(figsize=(20,10))
sns.barplot(x=data['Son'],y=data['Absent'],data=data)
```

Out[40]: <AxesSubplot:xlabel='Son', ylabel='Absent'>



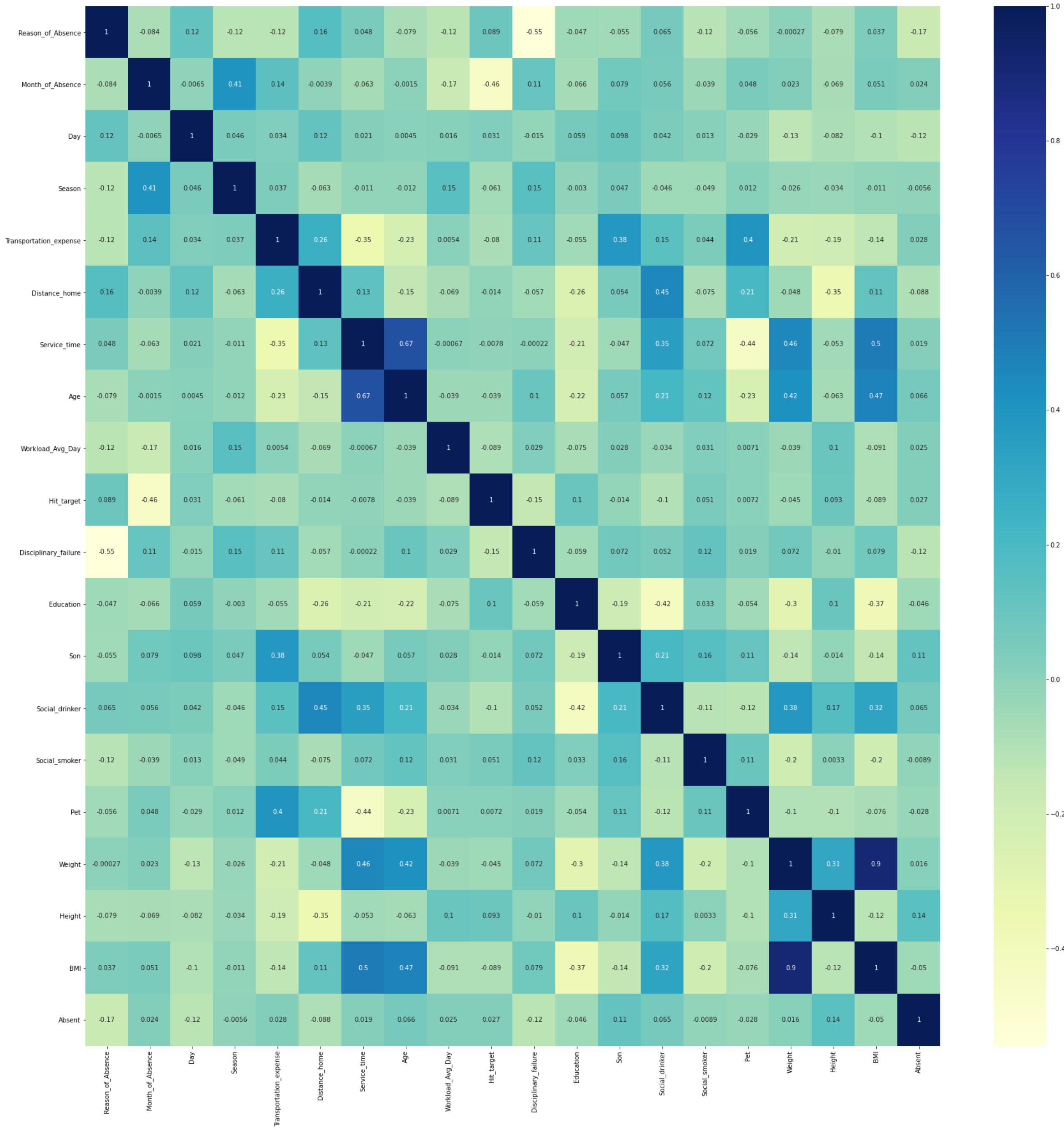
Agents having 3 kids tends to be absent for more hours at work.

In [107]:

```
plt.figure(figsize=(30,30))

sns.heatmap(data.corr(),cmap="YlGnBu",annot=True)
```

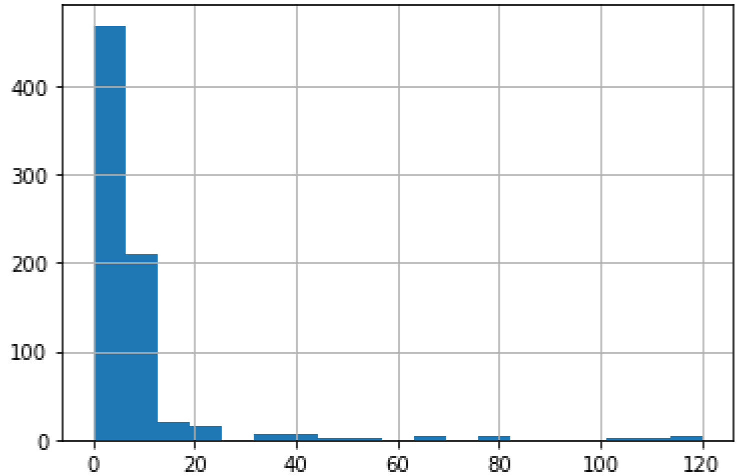
Out[107]: <AxesSubplot:>



Heatmap shows the correlation between the features. The weight and BMI feature is mostly correlated with the target feature.

In [110]:
```
data['Absent'].hist(bins=data['Absent'].nunique())
```

Out[110]: <AxesSubplot:>



Check where the Absent/ target variable lies, further we split it into categories for modelling.

In [114]:

```
data.columns
```

Out[114]:

```
Index(['Reason_of_Absence', 'Month_of_Absence', 'Day', 'Season',
       'Transportation_expense', 'Distance_home', 'Service_time', 'Age',
       'Workload_Avg_Day', 'Hit_target', 'Disciplinary_failure', 'Education',
       'Son', 'Social_drinker', 'Social_smoker', 'Pet', 'Weight', 'Height',
       'BMI', 'Absent'],
      dtype='object')
```

In [115]:

```
b=[]
for i in data['Absent']:
    if i>0:
        b.append(1)
    else:
        b.append(0)

data['Absent']=b



# Convert the Absent at work in hours data .
# If More than 0 consider absent and less than 0 not absent .
# As it will be easy for the target variable and modelling for classification problem.
```

In [44]:

```
x=data.drop('Absent',axis=1)


y=data['Absent']
print(y)


#Split the data in x ,y variables independent and dependent variables.
```

```
0      1
1      0
2      1
3      1
4      1
      ..
735    1
```

```
736    1
737    0
738    0
739    0
Name: Absent, Length: 740, dtype: int64
```

In [46]:

```
# Spliiting the data into train and test set.

from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=1)
```

In [48]:

```
print("Xtrain :",xtrain.shape)
print("ytrain :",ytrain.shape)
print("xtest :",xtest.shape)
print("ytest :",ytest.shape)


# Verify the size of the splitted data.
```

```
Xtrain : (592, 19)
ytrain : (592,)
xtest : (148, 19)
ytest : (148,)
```

Modelling

Decision Tree Classiifer

In [75]:

```
from sklearn import tree
t=tree.DecisionTreeClassifier()
t.fit(xtrain,ytrain)
ypred=t.predict(xtest)
acc_dt=t.score(xtest,ytest)*100

print("Accuracy :\n",acc_dt)
Dt_cm=metrics.confusion_matrix(ytest,ypred)
print("\nConfusion Matrix :\n",Dt_cm)


Classification_repo=metrics.classification_report(ytest,y_pred)
print('\n Classification Report :\n',Classification_repo)
```

```
Accuracy :
 99.32432432432432

Confusion Matrix :
 [[  5   1]
 [  0 142]]

 Classification Report :
               precision    recall  f1-score   support

           0       0.83      0.83      0.83         6
           1       0.99      0.99      0.99       142

    accuracy                           0.99       148
   macro avg       0.91      0.91      0.91       148
```

```
weighted avg       0.99      0.99      0.99       148
```

Logistic Regression

In [76]:

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
model_LR=LogisticRegression()
model_LR.fit(xtrain,ytrain)
y_pred2=model_LR.predict(xtest)
acc_LR=model_LR.score(xtest,ytest)*100
acc_LR


print("Accuracy :\n",acc_LR)
LR_cm=metrics.confusion_matrix(ytest,y_pred2)
print("\nConfusion Matrix :\n",LR_cm)

Classification_repo=metrics.classification_report(ytest,y_pred2)
print('\n Classification Report :\n',Classification_repo)
```

```
Accuracy :
 98.64864864864865

Confusion Matrix :
 [[  5   1]
 [  1 141]]

 Classification Report :
               precision    recall  f1-score   support

           0       0.83      0.83      0.83         6
           1       0.99      0.99      0.99       142

    accuracy                           0.99       148
   macro avg       0.91      0.91      0.91       148
weighted avg       0.99      0.99      0.99       148

C:\Users\asus\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Naive Bayes - Gaussian

In [77]:

```
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(xtrain,ytrain)
y_pred4=gnb.predict(xtest)
acc_gnb=gnb.score(xtest,ytest)*100

print("Accuracy :\n",acc_gnb)
gnb_cm=metrics.confusion_matrix(ytest,y_pred4)
print("\nConfusion Matrix :\n",gnb_cm)
```

```
Classification_repo=metrics.classification_report(ytest,y_pred4)
print('\n Classification Report :\n',Classification_repo)
```

```
Accuracy :
 98.64864864864865

Confusion Matrix :
 [[  4   2]
 [  0 142]]

 Classification Report :
               precision    recall  f1-score   support

           0       1.00      0.67      0.80         6
           1       0.99      1.00      0.99       142

    accuracy                           0.99       148
   macro avg       0.99      0.83      0.90       148
weighted avg       0.99      0.99      0.99       148
```

Random Forest Classifier

In [79]:

```
from sklearn.ensemble import RandomForestClassifier
Rf=RandomForestClassifier()
Rf.fit(xtrain,ytrain)
ypred3=Rf.predict(xtest)
Acc_Rf=Rf.score(xtest,ytest)*100
Acc_Rf


print("Accuracy :\n",Acc_Rf)
RF_cm=metrics.confusion_matrix(ytest,ypred3)
print("\nConfusion Matrix :\n",Dt_cm)
```

```
Accuracy :
 99.32432432432432

Confusion Matrix :
 [[  5   1]
 [  0 142]]
```

In [81]:

```
Classification_repo=metrics.classification_report(ytest,ypred3)
print('\n Classification Report :\n',Classification_repo)
```

```
 Classification Report :
               precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.99      1.00      1.00       142

    accuracy                           0.99       148
   macro avg       1.00      0.92      0.95       148
weighted avg       0.99      0.99      0.99       148
```

In [84]:

```
models=pd.DataFrame({
    'Model':['DecisionTree Classifier','Logistic Regression',
             'Random Forest',' Naive Bayes'],
    'Score':[acc_dt,acc_LR,Acc_Rf,acc_gnb]
})
```

```
models.sort_values(by='Score',ascending=False)
```

Out[84]:

	Model	Score
0	DecisionTree Classifier	99.324324
2	Random Forest	99.324324
1	Logistic Regression	98.648649
3	Naive Bayes	98.648649

The Highest Accuracy we got with Decision Tree Classifier and Random Forest Classifier.

In [ ]: