|  |
| --- |
| *# Importing Libraries* |
| import pandas as pd |
| import numpy as np |
| import matplotlib.pyplot as plt |
| import seaborn as sns |
| from scipy.stats import chi2\_contingency |
| from fancyimpute import KNN |
| import os |
| from sklearn.metrics import r2\_score |
| from scipy import stats |
| %matplotlib inline |

*# Setting working directory*

os.chdir("C:/Users/Ujjwal/Desktop/Project 1")

*# Loading data*

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

Exploratory Data Analysis

*# First 5 rows of data*

df.head()

|  |
| --- |
| *# Data Types of all the variables*  df.dtypes |
| Out: 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 |  |

*# Number of Unique values present in each variable*

df.nunique()

|  |  |  |
| --- | --- | --- |
| Out: 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 |  |

# df.shape

|  |  |
| --- | --- |
| Out: | (740, 21) |

*# From the EDA and problem statement file categorizing the variables in two category “Continues" and "Categorical"*

continues\_vars = ['Distance from Residence to Work', 'Service time', 'Age', 'Wo rk load Average/day ', 'Transportation expense','Hit target', 'Weight', 'Height', 'Body mass index', 'Absenteeism time inhours']

categorical\_vars = ['ID','Reason for absence','Month of absence','Day of the wee k','Seasons','Disciplinary failure', 'Education', 'Social drinker','Social smoker', 'Pet', 'Son']

Missing Value Analysis

*#Creating dataframe with missing values present in each variable*

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

*#Renaming variables of missing\_val dataframe*

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

*#Calculating percentage missing value*

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

*# Sorting missing\_val in Descending order*

missing\_val = missing\_val.sort\_values('Missing\_percentage', ascending = False).

reset\_index(drop = True)

*# Saving output result into csv file*

missing\_val.to\_csv("Missing\_perc.csv", index = False)

Imputation methods

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

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

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

(718, 21)

0

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

Out: 31.0

*# Checking for "Body mass index" column*

*# Actual value = 31*

*# Mean = 26.68*

*# Median = 25*

*# KNN = 30.99*

*# create missing value*

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

*# # Impute with mean*

* *df['Body mass index'] = df['Body mass index'].fillna(df['Body mass index'].mea n())*
* *df['Body mass index'].iloc[12]*
* *Impute with median*
* *df['Body mass index'] = df['Body mass index'].fillna(df['Body mass index'].med ian())*
* *df['Body mass index'].iloc[12]*

*# Apply KNN imputation algorithm*

df = pd.DataFrame(KNN(k = 3).complete(df), columns = df.columns)

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

Out: 30.999998833986012

In : *# Checking if all the missing value imputed* df.isnull().sum().sum()

Out: 0

OutLier Analysis

In : *# Ploting BoxPlot of continuous variables*

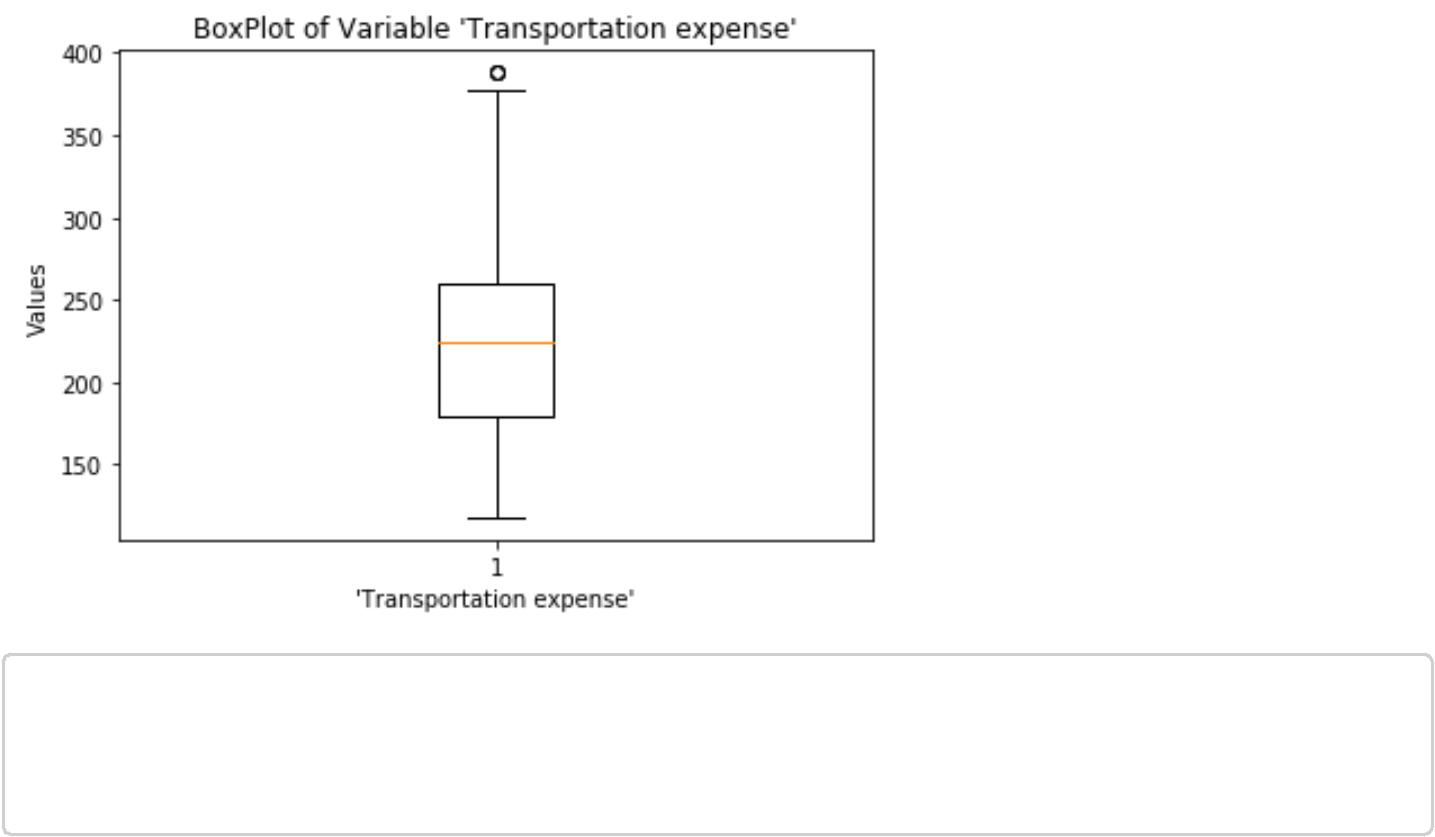
plt.boxplot(df['Transportation expense'])

plt.xlabel("'Transportation expense'")

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

plt.ylabel('Values')

Out: Text(0,0.5,'Values')



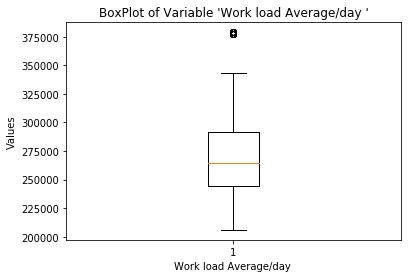
In : plt.boxplot(df['Work load Average/day '])

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

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

plt.ylabel('Values')

Out: Text(0,0.5,'Values')



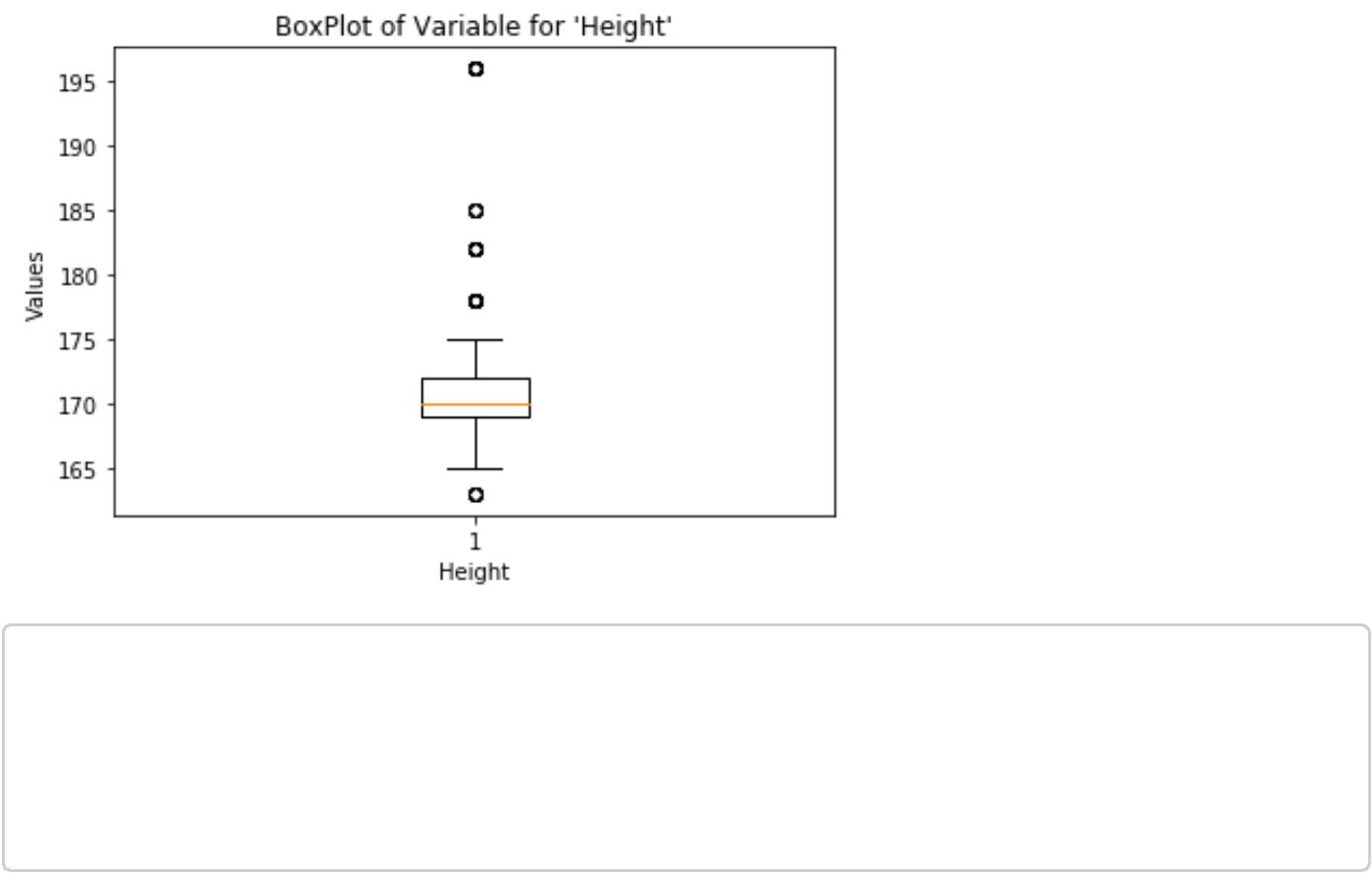
In : plt.boxplot(df['Height'])

plt.xlabel("Height")

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

plt.ylabel('Values')

Out: Text(0,0.5,'Values')



In : plt.boxplot([ df['Distance from Residence to Work'], df['Service time'], df['Ag

e'], df['Hit target'], df['Weight'], df['Body mass index']])

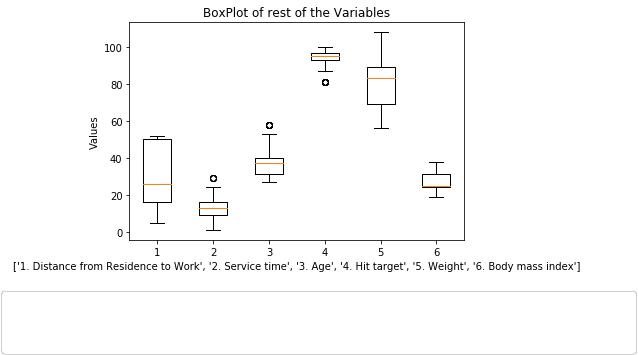
plt.xlabel(['1. Distance from Residence to Work', '2. Service time', '3. Age',

'4. Hit target', '5. Weight', '6. Body mass index'])

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

plt.ylabel('Values')

Out : Text(0,0.5,'Values')



In :*# From the above boxplot we can clearly see that in variables 'Distance from*

*Residence to Work', 'Weight' and 'Body mass index'*

*# there is no outlier*

*# list of variables which doesn’t have outlier*

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

* *Looping over all continues variables to detect and remove Outliers* for i in continues\_vars:
  + *Avoiding the variables which doesn't have outlier*

if i in neglect:

Continue

* *Getting 75 and 25 percentile of variable "i"* q75, q25 = np.percentile(df[i], [75,25])
* *Calculating Interquartile range*

iqr = q75 - q25

* + *Calculating upper extream and lower extream* minimum = q25 - (iqr\*1.5)

maximum = q75 + (iqr\*1.5)

* + *Replacing all the outliers value to NA* df.loc[df[i]< minimum,i] = np.nan df.loc[df[i]> maximum,i] = np.nan
* *Imputing missing values with KNN*

df = pd.DataFrame(KNN(k = 3).complete(df), columns = df.columns)

* *Checking if there is any missing value* df.isnull().sum().sum()

Imputing row 1/718 with 0 missing, elapsed time: 0.196

Imputing row 101/718 with 0 missing, elapsed time: 0.197

Imputing row 201/718 with 1 missing, elapsed time: 0.199

Imputing row 301/718 with 0 missing, elapsed time: 0.202

Imputing row 401/718 with 0 missing, elapsed time: 0.204

Imputing row 501/718 with 0 missing, elapsed time: 0.205

Imputing row 601/718 with 2 missing, elapsed time: 0.205

Imputing row 701/718 with 0 missing, elapsed time: 0.207

Out : 0

Feature Selection

In : *##Correlation analysis for continuous variables #Correlation plot*

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

In : *#Set the width and hieght of the plot*

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

*#Generate correlation matrix*

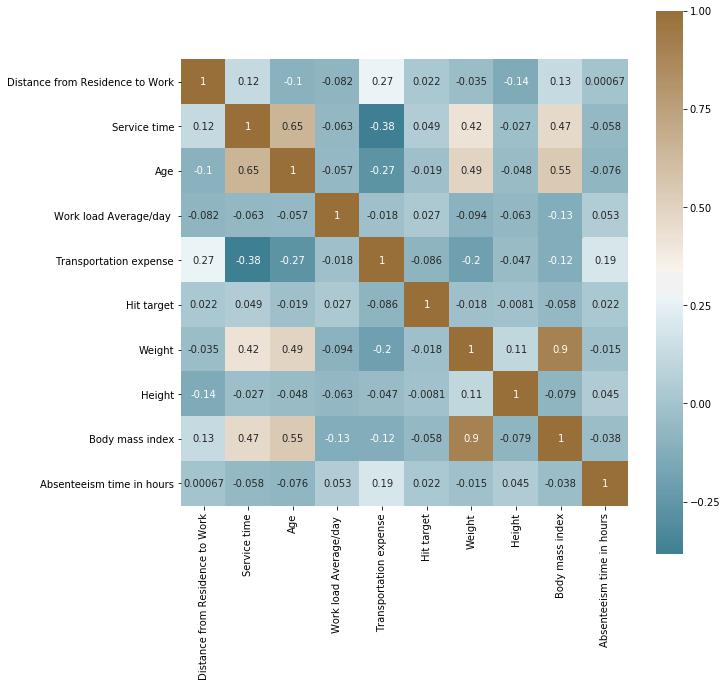
corr = df\_corr.corr()

*#Plot using seaborn library*

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool), cmap=sns.diverging\_palette(220, 50, as\_cmap=True), square=True, ax=ax, annot = True)

plt.plot()

Out : []



In : *#loop for ANOVA test Since the target variable is continuous* for i in categorical\_vars:

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

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

P value for variable ID is 8.449881295013552e-167

P value for variable Reason for absence is 9.770767089088417e-277

P value for variable Month of absence is 3.3124782278857673e-25

P value for variable Day of the week is 0.0008188161594849071

P value for variable Seasons is 3.127506937786291e-40

P value for variable Disciplinary failure is 1.2189432024253421e-185 P value for variable Education is 8.375003325123203e-105

P value for variable Social drinker is 1.2794395762714786e-150

P value for variable Social smoker is 9.117849965003895e-184

P value for variable Pet is 5.325984030592952e-127

P value for variable Son is 9.45269711512623e-116

In: *# Droping the variables which has redundant information* to\_drop = ['Weight']

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

In: *# Updating the Continuous Variables and Categorical Variables after droping some variables*

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

In: clean\_data = df.copy()

Feature Scaling:

In: *# Checking if there is any normally distributed variable in data* for i in continues\_vars:

if i == 'Absenteeism time in hours':

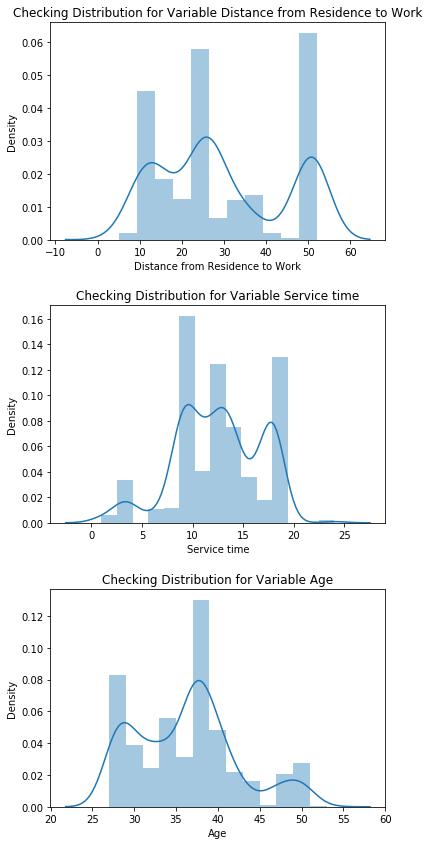
continue

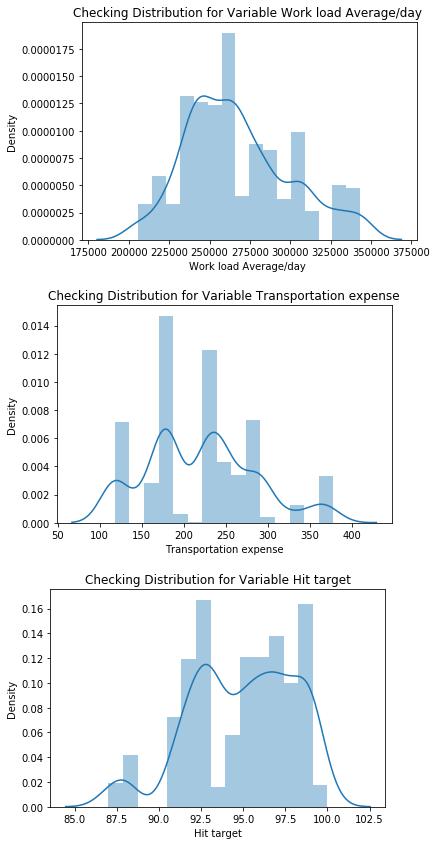
sns.distplot(df[i],bins = 'auto')

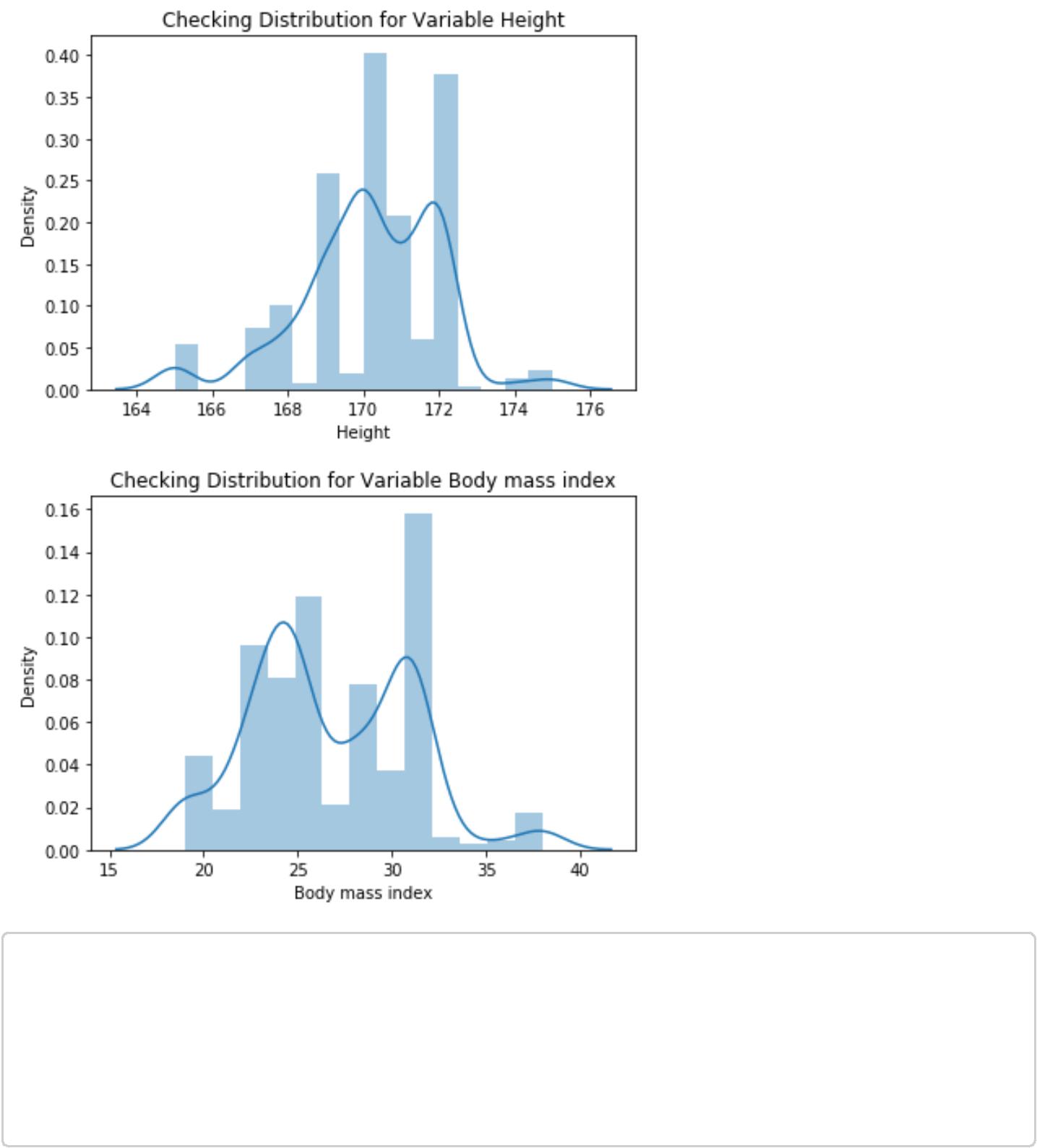
plt.title("Checking Distribution for Variable "+str(i))

plt.ylabel("Density")

plt.show()



**



In: *# Since there is no normally distributed curve we will use Normalizationg for Fe ature Scalling*

*# #Normalization*

for i in continues\_vars:

if i == 'Absenteeism time in hours':

continue

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

Machine Learning Models

In: *# Get dummy variables for categorical variables*

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

* *Copying dataframe* df1 = df.copy()

|  |  |  |
| --- | --- | --- |
| In: | df.iloc[:,8].head() | |
| Out: | 0 | 4.0 |

* 0.0
* 2.0
* 4.0

42.0

Name: Absenteeism time in hours, dtype: float64

In: df.iloc[:, df.columns != 'Absenteeism time in hours'].head(1)

In: *# Using train\_test\_split sampling function for test and train data split* from sklearn.model\_selection import train\_test\_split

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

Decision Tree

In: *# Importing libraries for Decision Tree*

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

*# Building model on top of training dataset*

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

* *Calculating RMSE for training data to check for over fitting* pred\_train = fit\_DT.predict(X\_train)

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

* *Calculating RMSE for test data to check accuracy*

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

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

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

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

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

)))

Root Mean Squared Error For Training data = 3.0750903832107097

Root Mean Squared Error For Test data = 3.3330757635157244

R^2 Score(coefficient of determination) = 0.09875989940860663

Random Forest

In: *# Importing libraries for Random Forest*

from sklearn.ensemble import RandomForestRegressor

*# Building model on top of training dataset*

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

*# Calculating RMSE for training data to check for over fitting* pred\_train = fit\_RF.predict(X\_train)

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

*# Calculating RMSE for test data to check accuracy* pred\_test = fit\_RF.predict(X\_test)

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

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

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

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

)))

Root Mean Squared Error For Training data = 0.9713567548389528

Root Mean Squared Error For Test data = 2.9293927985893746

R^2 Score(coefficient of determination) = 0.3038459480758552

Linear Regression

In: *# Importing libraries for Linear Regression*

from sklearn.linear\_model import LinearRegression

*# Building model on top of training dataset*

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

* *Calculating RMSE for training data to check for over fitting* pred\_train = fit\_LR.predict(X\_train)

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

* *Calculating RMSE for test data to check accuracy*

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

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

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

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

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

)))

Root Mean Squared Error For Training data = 2.2630941893032435

Root Mean Squared Error For Test data = 913872286061.1759

R^2 Score(coefficient of determination) = -6.7751848770029205e+22

Gradient Boosting

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
|  | # Importing Library for GradientBoosting  from sklearn.ensemble import GradientBoostingRegressor | | | |  |  |  |
|  | *#* | *Building model on* | *top* | *of training dataset* |

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

* *Calculating RMSE for training data to check for over fitting* pred\_train = fit\_GB.predict(X\_train)

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

* *Calculating RMSE for test data to check accuracy*

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

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

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

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

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

)))

Root Mean Squared Error For Training data = 1.9954754769178296

Root Mean Squared Error For Test data = 2.7532507448721244

R^2 Score(coefficient of determination) = 0.385047364820164

Dimensionality Reduction using PCA

In: target = df['Absenteeism time in hours']

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

df.shape

Out: (718, 129)

In: from sklearn.decomposition import PCA

* *Converting data to numpy array* X = df1.values
* *Data has 129 variables so no of components of PCA = 129*

pca = PCA(n\_components=129)

pca.fit(X)

* *The amount of variance that each PC explains*

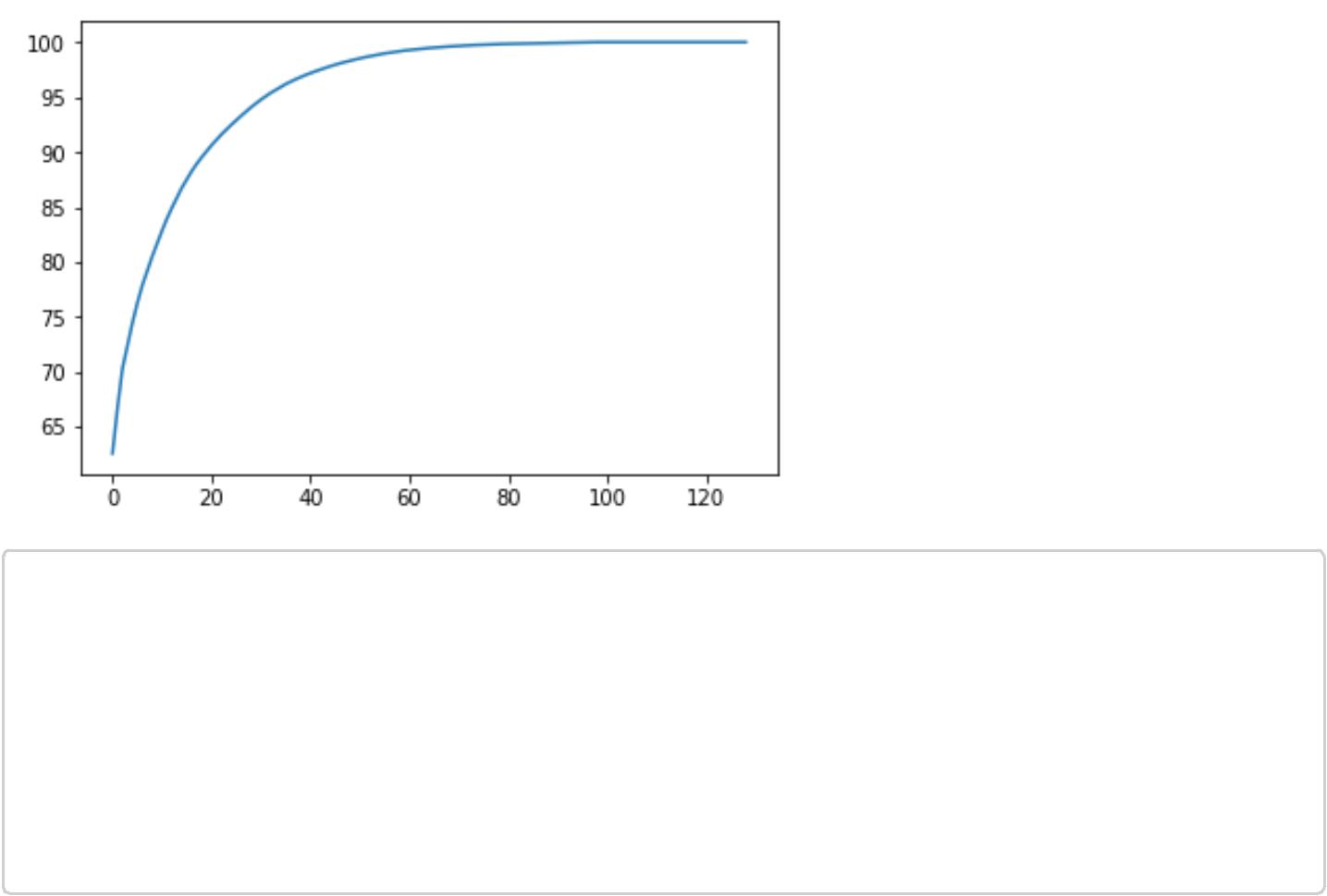
var= pca.explained\_variance\_ratio\_

*# Cumulative Variance explains*

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

plt.plot(var1)

plt.show()



In: *# From the above plot selecting 45 components since it explains almost 95+ % dat a variance*

pca = PCA(n\_components=45)

* *Fitting the selected components to the data* pca.fit(X)
* *Using train\_test\_split sampling function for test and train data split*

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

Decision Tree

In: *# Importing libraries for Decision Tree*

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

*# Building model on top of training dataset*

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

* *Calculating RMSE for training data to check for over fitting* pred\_train = fit\_DT.predict(X\_train)

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

* *Calculating RMSE for test data to check accuracy*

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

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

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

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

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

)))

Root Mean Squared Error For Training data = 0.5732916338845235

Root Mean Squared Error For Test data = 0.5666945950570649

R^2 Score(coefficient of determination) = 0.9719655423290273

Random Forest

In: *# Importing libraries for Random Forest*

from sklearn.ensemble import RandomForestRegressor

*# Building model on top of training dataset*

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

* *Calculating RMSE for training data to check for over fitting* pred\_train = fit\_RF.predict(X\_train)

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

* *Calculating RMSE for test data to check accuracy*

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

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

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

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

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

)))

Root Mean Squared Error For Training data = 0.033788086396731414

Root Mean Squared Error For Test data = 0.06772629598743314 R^2 Score(coefficient of determination) = 0.9995995865412352

Linear Regression

In: *# Importing libraries for Linear Regression*

from sklearn.linear\_model import LinearRegression

*# Building model on top of training dataset*

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

*# Calculating RMSE for training data to check for over fitting* pred\_train = fit\_LR.predict(X\_train)

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

*# Calculating RMSE for test data to check accuracy* pred\_test = fit\_LR.predict(X\_test)

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

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

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

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

)))

Root Mean Squared Error For Training data = 3.949343194785478e-15

Root Mean Squared Error For Test data = 0.0008015543803062408

R^2 Score(coefficient of determination) = 0.9999999439132853

Gradient Boosting

In: *# Importing library for Gradient Boosting*

from sklearn.ensemble import GradientBoostingRegressor

*# Building model on top of training dataset*

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

* *Calculating RMSE for training data to check for over fitting* pred\_train = fit\_GB.predict(X\_train)

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

* *Calculating RMSE for test data to check accuracy*

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

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

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

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

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

)))

Root Mean Squared Error For Training data = 0.0013224293438152876

Root Mean Squared Error For Test data = 0.02595486977005207

R^2 Score(coefficient of determination) = 0.999941192634185

Visual Analysis on Cleaned Data

In: from plotly.offline import download\_plotlyjs, init\_notebook\_mode, plot, iplot import cufflinks as cf

In: *# For Notebooks*

init\_notebook\_mode(connected=True)

In: cf.go\_offline()

Relationship of "Absenteeism time in hours" with others

In: *# Hist plot*

clean\_data.iplot(kind='hist',y='Absenteeism time in hours',bins='100')