**DATA SCIENCE COMPLETE ML LAB WITH PYTHON**

1. Documentation for individual task using R and Python
2. Documentation for packages used in R and Python
3. Documentation for data partition using R and Python
4. Documentation for syntax of model building in R and Python
5. Documentation for syntax for predicting using R and Python
6. Documentation of syntax for accuracy score in R and Python
7. Wbcd data set apply all algorithms on this dataset using R and Python

**Packages in Python:**

1. **Upload the dataset using Pandas**

Packages:

**import pandas as pd**

1. **Pre-Processing the data using Numpy,Scikit**

Packages:

**import numpy as np**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.preprocessing import MinMaxScaler**

**from sklearn.preprocessing import RobustScaler**

**from sklearn.preprocessing import PolynomialFeatures**

**from sklearn.preprocessing import OneHotEncoder**

**from sklearn.preprocessing import Binarizer**

**from sklearn.preprocessing import FunctionTransformer**

**from sklearn.preprocessing import Imputer**

1. **Exploratory Data Analysis (EDA)**

Packages:

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**import statistics**

**from scipy.stats import skew , kurtosis**

1. **Data Partition**

Packages:

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

**from sklearn.cross\_validation import train\_test\_split**

1. **Un-Supervised Learning**

Packages:

**from scipy.cluster.hierarchy import dendrogram, linkage**

**from sklearn.cluster import AgglomerativeClustering**

**import scipy.cluster.hierarchy as shc**

**from scipy.spatial.distance import cdist**

**from sklearn.decomposition import PCA**

**from sklearn.cluster import KMeans**

**from scipy.spatial import distance**

1. **Supervised Learning**

Packages:

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.neighbors import KNeighborsClassifier**

**from sklearn.naive\_bayes import GaussianNB**

**from sklearn.svm import SVC**

1. **Confusion Matrix**

Packages:

**from sklearn.metrics import confusion\_matrix**

1. **Accuracy**

Packages:

**from sklearn.metrics import accuracy\_score**

**ML Supervised Learning Algorithms**

1. **K-Nearest Neighbor (KNN)**
2. **Naïve Bayes Classifier (NBC)**
3. **Support Vector Machine (SVM)**
4. **Decision Trees**
5. **Random Forest**
6. **KNN:**

# Upload the dataset using Pandas

**import pandas as pd**

# Upload the wbcd dataset

**wbcd = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - R\\KNN\\wbcd.csv")**

**wbcd**

# Delete the unwanted from dataset

**wbcd1=wbcd.drop(columns='id')**

**wbcd1**

# Divide the data as input and output

**x\_input = wbcd1.iloc[: , 1:].values**

**x\_input**

**y\_output = wbcd1.iloc[:,0].values**

**y\_output**

# Split the data

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

**x\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_input ,y\_output,train\_size=0.80)**

# Build the KNN Model

**from sklearn.neighbors import KNeighborsClassifier**

**classifier = KNeighborsClassifier(n\_neighbors=3)**

**classifier.fit(x\_train,y\_train)**

# Predict the Model

**y\_pred =classifier.predict(x\_test)**

**y\_pred**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_pred,y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy =accuracy\_score(y\_pred,y\_test)**

**accuracy**

1. **Naïve Bayes Classifier**

# Import dataset using Pandas

**import pandas as pd**

# Upload the data set

**wbcd = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - R\\KNN\\wbcd.csv")**

**wbcd**

# Delete the unwanted data from dataset

**wbcd1=wbcd.drop(columns='id')**

**wbcd1**

# Divide the data as input and output

**x\_input =wbcd1.iloc[:, 1:].values**

**print("input values:",x\_input)**

**y\_output =wbcd1.iloc[:,0].values**

**print("output values:",y\_output)**

# Split the data

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

**x\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_input,y\_output,train\_size=0.80)**

# Build the NBC Model

**from sklearn.naive\_bayes import GaussianNB**

**model = GaussianNB()**

**model.fit(x\_train,y\_train)**

# Predict the model

**y\_pred =model.predict(x\_test)**

**y\_pred**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_pred,y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy =accuracy\_score(y\_pred,y\_test)**

**accuracy**

1. **SVM**

# Import packages for dataset upload

**import pandas as pd**

# Upload the WBCD dataset

**wbcd = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - R\\KNN\\wbcd.csv")**

**wbcd**

# Delete the unwanted data drom dataset

**wbcd1=wbcd.drop(columns='id')**

**wbcd1**

# Divide the data as input and output

**x\_input = wbcd1.iloc[:,1:].values**

**print("input values:",x\_input)**

**y\_output = wbcd1.iloc[:,0].values**

**print("output values:",y\_output)**

# Split the data as train and test

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

**x\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_input,y\_output ,train\_size=0.80)**

# Build the SVM model

**from sklearn.svm import SVC**

**svm\_model = SVC(kernel='poly' ,random\_state=0)**

**svm\_model.fit(x\_train,y\_train)**

# Predict the model

**svm\_pred = svm\_model.predict(x\_test)**

**svm\_pred**

# Confusion matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(svm\_pred,y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**svm\_accuracy=accuracy\_score(svm\_pred,y\_test)**

**svm\_accuracy**

1. **Decision Tree**

# upload the pandas

**import pandas as pd**

# upload the wbcd dataset

**wbcd = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - R\\KNN\\wbcd.csv")**

**wbcd**

# Delete the unwanted data from dataset

**wbcd1=wbcd.drop(columns='id')**

**wbcd1**

# Divide the data as input and output

**x\_input = wbcd1.iloc[:,1:].values**

**print("input values:",x\_input)**

**y\_output = wbcd1.iloc[:,0].values**

**print("output values:",y\_output)**

# Split the data as train and test

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

**x\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_input,y\_output ,train\_size=0.80)**

# Build Decision Trees

**from sklearn.tree import DecisionTreeClassifier**

**classifier=DecisionTreeClassifier(criterion='entropy',random\_state=0)**

**classifier.fit(x\_train,y\_train)**

# Predict Model

**y\_predict = classifier.predict(x\_test)**

**y\_predict**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_predict,y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy=accuracy\_score(y\_predict,y\_test)**

**accuracy**

1. **Random Forest**

# upload the pandas

**import pandas as pd**

# upload the wbcd dataset

**wbcd = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - R\\KNN\\wbcd.csv")**

**wbcd**

# Delete the unwanted data from dataset

**wbcd1=wbcd.drop(columns='id')**

**wbcd1**

# Divide the data as input and output

**x\_input = wbcd1.iloc[:,1:].values**

**print("input values:",x\_input)**

**y\_output = wbcd1.iloc[:,0].values**

**print("output values:",y\_output)**

# Split the data as train and test

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

**x\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_input,y\_output ,train\_size=0.80)**

# Build Random Forest Model

**from sklearn.ensemble import RandomForestClassifier**

**classifier=RandomForestClassifier(criterion='entropy',random\_state=0)**

**classifier.fit(x\_train,y\_train)**

# Predict Model

**y\_predict = classifier.predict(x\_test)**

**y\_predict**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_predict,y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy=accuracy\_score(y\_predict,y\_test)**

**accuracy**

**ML Un-Supervised Learning Algorithms**

1. **H-Clustering**
2. **K-Means Clustering**
3. **Principal Component Analysis(PCA)**
4. **PCA:**

# import the pandas package

**import pandas as pd**

# Upload wbcd dataset

**wbcd = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - R\\KNN\\wbcd.csv")**

**wbcd**

# delete the unwanted data from dataset

**wbcd1 = wbcd.drop(columns='id')**

**wbcd1**

# delete another unwanted data

**wbcd2 = wbcd1.drop(columns='diagnosis')**

**wbcd2**

# Normalize the data

**from sklearn.preprocessing import scale**

**wbcd2\_norm = scale(wbcd2)**

**wbcd2\_norm**

# find the no.of columns and rows in the dataset

**wbcd2.shape**

# Apply the PCA model

**from sklearn.decomposition import PCA**

**pca=PCA(n\_components=12)**

**pca**

# fit the model

**pca\_values = pca.fit\_transform(wbcd2\_norm)**

**pca\_values**

# The amount of variance that each PCA explains is

**var = pca.explained\_variance\_ratio\_**

**var**

# converting into exact percentage using numpy

**import numpy as np**

**var1 = np.cumsum(np.round(var,decimals = 4)\*100)**

**var1**

# Calculate Principal component

**principalDf = pd.DataFrame(data = pca\_values,columns = ['principal component 1',**

**'principal component 2','principal component 3','principal component**

**4','principal component 5', 'principal component 6','principal component**

**7','principal component 8','principal component 9','principal component**

**10','principal component 11', 'principal component 12'])**

**principalDf**

1. **H-Clustering :**

# Upload required packages

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**%matplotlib inline**

# Upload the dataset

**data = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - Python\\Wholesale customers data.csv")**

**Data**

# first normalize the data and bring all the variables to the same scale

**from sklearn.preprocessing import normalize**

**data\_scaled = normalize(data)**

**data\_scaled = pd.DataFrame(data\_scaled, columns=data.columns)**

**data\_scaled.head(10)**

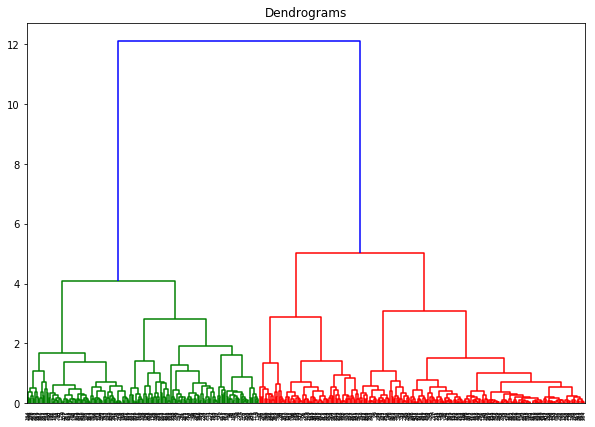
# first draw the dendrogram to help us decide the number of clusters for this particular problem

**import scipy.cluster.hierarchy as shc**

**plt.figure(figsize=(10, 7))**

**plt.title("Dendrograms")**

**dend = shc.dendrogram(shc.linkage(data\_scaled, method='ward'))**

****

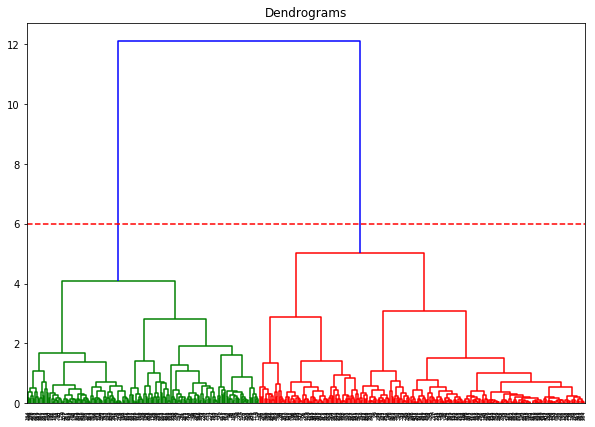
# The x-axis contains the samples and y-axis represents the distance between these samples. The vertical line with maximum distance is the blue line and hence we can decide a threshold of 6 and cut the dendrogram**:**

**plt.figure(figsize=(10, 7))**

**plt.title("Dendrograms")**

**dend = shc.dendrogram(shc.linkage(data\_scaled, method='ward'))**

**plt.axhline(y=6, color='r', linestyle='--')**

****

# We have two clusters as this line cuts the dendrogram at two points. Let’s now apply hierarchical clustering for 2 clusters

**from sklearn.cluster import AgglomerativeClustering**

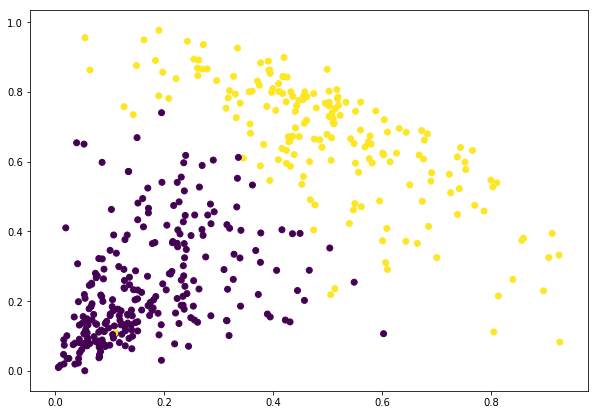
**cluster = AgglomerativeClustering(n\_clusters=2, affinity='euclidean', linkage='ward')**

**cluster.fit\_predict(data\_scaled)**

# We can see the values of 0s and 1s in the output since we defined 2 clusters. 0 represents the points that belong to the first cluster and 1 represents points in the second cluster. Let’s now visualize the two clusters

**plt.figure(figsize=(10, 7))**

**plt.scatter(data\_scaled['Milk'], data\_scaled['Grocery'], c=cluster.labels\_)**

****

1. **K- Means Clustering:**

# upload required packages

**import numpy as np**

**import pandas as pd**

**import matplotlib.pylab as plt**

**from sklearn.cluster import KMeans**

**from scipy.spatial.distance import cdist**

# upload the dataset

**data = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - Python\\Wholesale customers data.csv")**

**data**

# Normalize the data

**from sklearn.preprocessing import normalize**

**data\_scale = normalize(data)**

**data\_scale=pd.DataFrame(data\_scale, columns=data.columns)**

**data\_scale.head()**

# apply K-means clustering

**wss = []**

#Sum\_of\_squared\_distances = []

**K = np.arange(1,15)**

**for k in K:**

**km = KMeans(n\_clusters=k)**

**km = km.fit(data\_scale)**

**wss.append(km.inertia\_)**

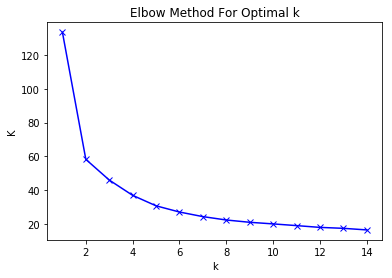
**plt.plot(K, wss, 'bx-')**

**plt.xlabel('k')**

**plt.ylabel(K)**

**plt.title('Elbow Method For Optimal k')**

**plt.show()**

****

# Apply K-means

**model=KMeans(n\_clusters=5)**

**model**

# Fit the model

**abc = model.fit(data\_scale)**

**abc**

# Predict the model

**from sklearn.cluster import KMeans**

**y\_kmeans = model.predict(data\_scale)**

**y\_kmeans**

# Add the new column to existing

**data['clusters']=pd.Series(y\_kmeans)**

**data**

**Apply all ML Algorithms on one Dataset**

**Upload the wbcd dataset for Analysis**

# Upload required packages

**import pandas as pd**

# upload the dataset

**wbcd = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - Python\\wbcd.csv")**

**wbcd**

# Check if any missing values in Dataset

# Check any missing values in the dataset

**wbcd.apply(lambda wbcd: sum(wbcd.isnull()),axis=0)**

# Delete unwanted data from Dataset

# Delete unwanted data from Dataset

**wbcd1 = wbcd.drop(columns='id')**

**wbcd1**

# Divide the dataset into Input and Output Values

# Divide the dataset as output

**wbcd1\_output=wbcd1.iloc[:,0]**

**wbcd1\_output**

# Divide the dataset as input

**wbcd1\_input=wbcd1.iloc[:,1:]**

**wbcd1\_input**

# Normalize / Scale the data

# Normalize the data

**# from sklearn.preprocessing import normalize**

**# wbcd1\_norm =normalize(wbcd1\_input)**

**# wbcd1\_norm**

# Scaling the data

**from sklearn.preprocessing import scale**

**wbcd1\_scale = scale(wbcd1\_input)**

**wbcd1\_scale**

# Apply Principal Component Analysis

# Apply PCA Algorithm

**from sklearn.decomposition import PCA**

**pca=PCA(n\_components=13)**

**pca**

# fit the model using scale

**pca\_values = pca.fit\_transform(wbcd1\_scale)**

**pca\_values**

# The amount of variance that each PCA explains is

**var = pca.explained\_variance\_ratio\_**

**var**

# converting into exact percentage using numpy

**import numpy as np**

**var1 = np.cumsum(np.round(var,decimals = 4)\*100)**

**var1**

# now we will take only 13 columns for data analysis

**wbcd1\_input = wbcd1.iloc[: , 1:13]**

**wbcd1\_input**

# Exploratory Data Analysis (EDA)

# perform EDA on Input data

**wbcd1\_input.describe()**

# Transpose the data

**wbcd1\_input.describe().transpose()**

# perform column count on Output data

**wbcd1\_output.value\_counts()**

# Calculate SKewness and kurtosis

**import matplotlib.pyplot as plt**

**import seaborn as sns**

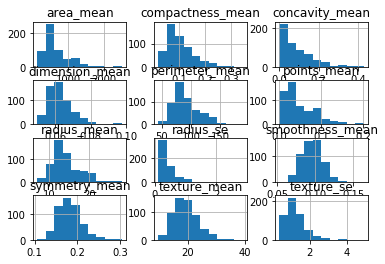
**from scipy.stats import skew , kurtosis**

**print("Skewness of data" , skew(wbcd1\_input))**

**print("Kurtosis of data" , kurtosis(wbcd1\_input))**

# Various graphical representations

**wbcd1\_input.hist()**

****

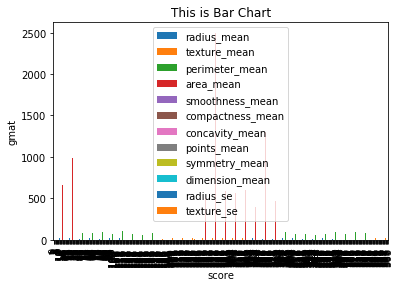
# Bar Plot

**wbcd1\_input.plot(kind='bar')**

**plt.xlabel("score")**

**plt.ylabel("gmat")**

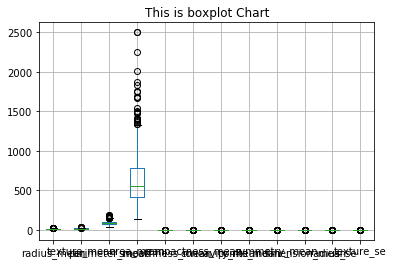
**plt.title("This is Bar Chart")**

****

# Boxplot chart

**wbcd1\_input.boxplot()**

**plt.title("This is boxplot Chart")**

****

# Fill the missing values if available in dataset

# Filling the missing values in particular column

**# df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)**

# Split the data into Training and Testing

# Split the data as training and testing

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

**x\_train,x\_test,y\_train,y\_test = train\_test\_split(wbcd1\_input,wbcd1\_output,train\_size =0.80)**

# Apply Supervised Learning Algorithms

# KNN

# Build KNN Algorithms

**from sklearn.neighbors import KNeighborsClassifier**

**classifier = KNeighborsClassifier(n\_neighbors=3)**

**classifier.fit(x\_train,y\_train)**

# Predict the Model

**y\_pred =classifier.predict(x\_test)**

**y\_pred**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_pred,y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy =accuracy\_score(y\_pred,y\_test)**

**accuracy**

# Decision Tree

# 2. Decision Tree

**from sklearn.tree import DecisionTreeClassifier**

**classifier1=DecisionTreeClassifier(criterion='entropy',random\_state=0)**

**classifier1.fit(x\_train,y\_train)**

# Predict Model

**y\_predict1 = classifier1.predict(x\_test)**

**y\_predict1**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_predict1,y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy1=accuracy\_score(y\_predict1,y\_test)**

**accuracy1**

# Random Forest

# 3. Random Forest

**from sklearn.ensemble import RandomForestClassifier**

**classifier2=RandomForestClassifier(criterion='entropy',random\_state=0)**

**classifier2.fit(x\_train,y\_train)**

# Predict Model

**y\_predict2 = classifier2.predict(x\_test)**

**y\_predict2**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_predict2,y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy2=accuracy\_score(y\_predict2,y\_test)**

**accuracy2**

# Naive Bayes Classifier (NBC)

# 4. Naive Bayes Classifier (NBC)

**from sklearn.naive\_bayes import GaussianNB**

**classifier3 = GaussianNB()**

**classifier3.fit(x\_train,y\_train)**

# Predict the model

**y\_predict3 = classifier3.predict(x\_test)**

**y\_predict3**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_predict3,y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy3=accuracy\_score(y\_predict3,y\_test)**

**accuracy3**

**IRIS Dataset**

# Upload the IRIS dataset for Analysis

# Upload Pandas Package

**import pandas as pd**

# Upload the IRIS dataset

**iris = pd.read\_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\Social Prachar**

**Material\\Codes With Examples - Python\\iris.csv")**

**Iris**

# Check rows and columns size

**print("Dimension of the data :",iris.ndim)**

**print("shape of the data : ",iris.shape)**

# Divide the dataset into input and output

# Divide the data as input

**iris\_input = iris.iloc[: , 0:4]**

**iris\_input**

# Divide the data as output

**iris\_output = iris.iloc[: ,4]**

**iris\_output**

# Check if any missing values in data set

# Missing values finding

**iris.apply(lambda iris: sum(iris.isnull()),axis=0)**

# Perform Exploratory Data Analysis (EDA)

# First ,second Business Rules

**iris\_input.describe()**

# Transpose the data

**iris\_input.describe().transpose()**

# Caluculate columns count on Discrete variables output data

**iris\_output.value\_counts()**

# Find Skewness

**from scipy.stats import skew , kurtosis**

**skew(iris\_input)**

# Find Kurtosis

**kurtosis(iris\_input)**

# Various Graphical representation

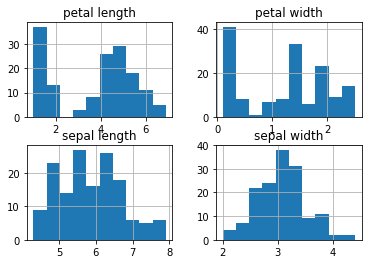
# Histogram

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**iris\_input.hist()**

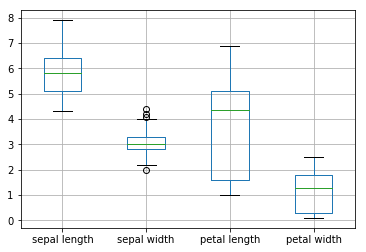
**plt.show()**

****

# Boxplot

**iris\_input.boxplot()**

**plt.show()**

****

# Scale / Normalize the data

# Scale the data

**from sklearn.preprocessing import scale**

**iris\_scale = scale(iris\_input)**

**iris\_scale**

# Split the data into training and testing

# partition the data based on setosa

**iris\_setosa = iris.iloc[0:50 , 0:5]**

**iris\_setosa**

# partition the data based on versicolor

**iris\_versicolor = iris.iloc[50:100 , 0:5]**

**iris\_versicolor**

# partition the data based on virginica

**iris\_virginica = iris.iloc[100:150 , 0:5]**

**iris\_virginica**

# Club the data from individually on 25% from each as training dataset

**df1 = pd.DataFrame(iris\_setosa.iloc[0:25 , 0:5])**

**df1**

**df2 = pd.DataFrame(iris\_versicolor.iloc[0:25 , 0:5])**

**df2**

**df3 = pd.DataFrame(iris\_virginica.iloc[0:25 , 0:5])**

**df3**

# Concatenate all three dataframes into Training Dataset

**iris\_train = pd.concat([df1,df2,df3])**

**iris\_train**

# shape of the training dataset

**iris\_train.shape**

# Club the data from individually on 25% from each as training dataset

**df4 = pd.DataFrame(iris\_setosa.iloc[25:50 , 0:5])**

**df4**

**df5 = pd.DataFrame(iris\_versicolor.iloc[25:50 , 0:5])**

**df5**

**df6 = pd.DataFrame(iris\_virginica.iloc[25:50 , 0:5])**

**df6**

# Concatenate all three dataframes into Testing Dataset

**iris\_test = pd.concat([df4,df5,df6])**

**iris\_test**

# shape of the testing dataset

**iris\_test.shape**

# Dividing the input and output variable for Training Dataset

**X\_train = iris\_train.iloc[:,0:4]**

**X\_train**

**Y\_train = iris\_train.iloc[:,4]**

**Y\_train**

# Dividing the input and output variable for Training Dataset

**X\_test = iris\_test.iloc[:,0:4]**

**X\_test**

**Y\_test = iris\_test.iloc[:,4]**

**Y\_test**

# Apply All ML algorithms

# Support Vector Machine (SVM)

**from sklearn.svm import SVC**

**classifier = SVC(kernel = 'linear', random\_state = 0)**

**classifier.fit(X\_train,Y\_train)**

# Predicting the Test set results

**y\_pred = classifier.predict(X\_test)**

**y\_pred**

# Making the Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(Y\_test, y\_pred))**

# accuracy

**from sklearn.metrics import accuracy\_score**

**Accuracy = accuracy\_score(Y\_test, y\_pred)**

**Accuracy**

# K-Nearest Neighbor

**from sklearn.neighbors import KNeighborsClassifier**

**classifier1 = KNeighborsClassifier(n\_neighbors=4)**

**classifier1.fit(X\_train,Y\_train)**

# Predict the model

**y\_pred1 = classifier1.predict(X\_test)**

**y\_pred1**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_pred1, Y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy1=accuracy\_score(y\_pred1,Y\_test)**

**accuracy1**

# Naive Bayes Classifier

**from sklearn.naive\_bayes import GaussianNB**

**classifier2=GaussianNB()**

**classifier2.fit(X\_train,Y\_train)**

# Predict the model

**y\_pred2 = classifier2.predict(X\_test)**

**y\_pred2**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_pred2 , Y\_test))**

# accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy2 = accuracy\_score(y\_pred2 , Y\_test)**

**accuracy2**

# Decision Tree

**from sklearn.tree import DecisionTreeClassifier**

**classifier3 = DecisionTreeClassifier(criterion ='entropy' ,random\_state=0)**

**classifier3.fit(X\_train , Y\_train)**

# Predict the model

**y\_pred3 = classifier3.predict(X\_test)**

**y\_pred3**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_pred3,Y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy3 = accuracy\_score(y\_pred3 , Y\_test)**

**accuracy3**

# Random Forest

**from sklearn.ensemble import RandomForestClassifier**

**classifier4 = RandomForestClassifier(criterion ='entropy',random\_state=0)**

**classifier4.fit(X\_train,Y\_train)**

# predict the model

**y\_pred4 = classifier4.predict(X\_test)**

**y\_pred4**

# Confusion Matrix

**from sklearn.metrics import confusion\_matrix**

**print(confusion\_matrix(y\_pred4 ,Y\_test))**

# Accuracy

**from sklearn.metrics import accuracy\_score**

**accuracy4= accuracy\_score(y\_pred4, Y\_test)**

**accuracy4**