**Random Forest**

A Random Forest is an ensemble technique(**Ensemble methods** is a machine learning technique that combines several base models in order to produce one optimal predictive model) capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as **bagging**.

**Ensemble methods** is a machine learning technique that combines several base models in order to produce one optimal predictive model.

The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.  
Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

We need to approach the Random Forest regression technique like any other machine learning technique

* Design a specific question or data and get the source to determine the required data.
* Make sure the data is in an accessible format else convert it to the required format.
* Specify all noticeable anomalies and missing data points that may be required to achieve the required data.
* Create a machine learning model
* Set the baseline model that you want to achieve
* Train the data machine learning model.
* Provide an insight into the model with test data
* Now compare the performance metrics of both the test data and the predicted data from the model.
* If it doesn’t satisfy your expectations, you can try improving your model accordingly or dating your data or use another data modeling technique.
* At this stage you interpret the data you have gained and report accordingly.

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn’t depend on one decision tree but multiple decision trees. In the case of a classification problem, the final output is taken by using the majority voting classifier. In the case of a regression problem, the final output is the mean of all the outputs. This part is Aggregation.

Problem definition:

The data is related with direct marketing campaigns of a banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) was subscribed or not. Data set has 17 predictor variables (features) and around 45K rows.

# Creation of base dataset

import numpy as np

import pandas as pd

data\_term\_dep = pd.read\_csv("bank\_customer\_survey.csv")

*#Top 5 rows of the dataset is shown below.*

data\_term\_dep.head()

*#Dataset has 17columns and around 45k rows as shown below:*

data\_term\_dep.shape

## **Null value treatment**

In [3]:

def null\_values(base\_dataset):

print(base\_dataset.isna().sum())

*## null value percentage*

null\_value\_table=(base\_dataset.isna().sum()/base\_dataset.shape[0])\*100

*## null value percentage beyond threshold drop , else treat the columns*

retained\_columns=null\_value\_table[null\_value\_table<30].index

*# if any variable as null value greater than input(like 30% of the data) value than those variable are consider as drop*

drop\_columns=null\_value\_table[null\_value\_table>30].index

base\_dataset.drop(drop\_columns,axis=1,inplace=True)

len(base\_dataset.isna().sum().index)

cont=base\_dataset.describe().columns

cat=[i for i **in** base\_dataset.columns if i **not** **in** base\_dataset.describe().columns]

for i **in** cat:

base\_dataset[i].fillna(base\_dataset[i].value\_counts().index[0],inplace=True)

for i **in** cont:

base\_dataset[i].fillna(base\_dataset[i].median(),inplace=True)

print(base\_dataset.isna().sum())

return base\_dataset,cat,cont

In [4]:

data\_term\_dep2,cat,cont=null\_values(data\_term\_dep1)

## **Outlier treatment**

In [5]:

def outliers\_transform(base\_dataset):

for i **in** base\_dataset.var().sort\_values(ascending=False).index[0:10]:

x=np.array(base\_dataset[i])

qr1=np.quantile(x,0.25)

qr3=np.quantile(x,0.75)

iqr=qr3-qr1

utv=qr3+(1.5\*(iqr))

ltv=qr1-(1.5\*(iqr))

y=[]

for p **in** x:

if p <ltv **or** p>utv:

y.append(np.median(x))

else:

y.append(p)

base\_dataset[i]=y

In [6]:

outliers\_transform(data\_term\_dep2)

In [7]:

*#Display the columns after outlier treatment*

data\_term\_dep2.columns

## **Dummy Variable Declaration**

In [8]:

dummy\_columns=[]

for i **in** data\_term\_dep2.columns:

if (data\_term\_dep2[i].nunique()>=3) & (data\_term\_dep2[i].nunique()<5):

dummy\_columns.append(i)

In [9]:

dummy\_columns

In [10]:

*#Dummy Variable*

dummies\_tables=pd.get\_dummies(data\_term\_dep2[dummy\_columns])

In [11]:

for i **in** dummies\_tables.columns:

data\_term\_dep2[i]=dummies\_tables[i]

In [12]:

*#Displaying columns after dummy variable creation*

data\_term\_dep2.columns

In [13]:

*#Drop the existing columns after the creation of dummy variable for those*

data\_term\_dep2=data\_term\_dep2.drop(dummy\_columns,axis=1)

In [14]:

data\_term\_dep2.columns

Label Encoder

In [15]:

from sklearn.preprocessing import LabelEncoder

def label\_encoders(data,cat):

le=LabelEncoder()

for i **in** cat:

le.fit(data[i])

x=le.transform(data[i])

data[i]=x

return data

In [16]:

data\_new=data\_term\_dep2

cat=data\_term\_dep2.describe(include='object').columns

In [17]:

label\_encoders(data\_new,cat).head()

In [18]:

data\_new.columns

## **Univariate analysis (EDA)**

In [19]:

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))

for i **in** data\_new.var().index:

sns.distplot(data\_new[i],kde=False)

plt.show()

## **Bivariate analysis (EDA)**

In [20]:

plt.figure(figsize=(20,10))

sns.heatmap(data\_new.corr())

# Model Building

## **Supervised**

### **Regression**

### **Classification**

In [21]:

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import BaggingClassifier

In [22]:

y=data\_new['y']

x=data\_new.drop('y',axis=1)

In [23]:

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

X\_train,X\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.20,random\_state=43)

In [24]:

print(X\_train.shape,X\_test.shape,y\_train.shape,y\_test.shape)

In [25]:

models=[DecisionTreeClassifier(),RandomForestClassifier(),BaggingClassifier()]

In [26]:

from sklearn.metrics import confusion\_matrix,accuracy\_score

final\_accuracy\_scores=[]

for i **in** models:

dt=i

dt.fit(X\_train,y\_train)

dt.predict(X\_test)

dt.predict(X\_train)

print(confusion\_matrix(y\_test,dt.predict(X\_test)))

print(accuracy\_score(y\_test,dt.predict(X\_test)))

print(confusion\_matrix(y\_train,dt.predict(X\_train)))

print(accuracy\_score(y\_train,dt.predict(X\_train)))

print(i)

final\_accuracy\_scores.append([i,confusion\_matrix(y\_test,dt.predict(X\_test)),accuracy\_score(y\_test,dt.predict(X\_test)),confusion\_matrix(y\_train,dt.predict(X\_train)),accuracy\_score(y\_train,dt.predict(X\_train))])

from sklearn.model\_selection import cross\_val\_score

print(cross\_val\_score(i,X\_train,y\_train,cv=10))

In [27]:

final\_accuracy\_scores1=pd.DataFrame(final\_accuracy\_scores)

In [28]:

final\_accuracy\_scores1

In [29]:

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test,dt.predict(X\_test))