**INTRODUCTION :**

We are using XGboost regression technique. It's a type of ensemble learning where we can optimize the hyperparameters.

**DATASET :**

We are using the Predicting House Prices data set . train1 contains training data.

Columns - age: age of primary beneficiary

* sex: insurance contractor gender, female, male
* bmi: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height, objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9
* children: Number of children covered by health insurance / Number of dependents
* smoker: Smoking
* region: the beneficiary's residential area in the US, northeast, southeast, southwest, northwest.
* charges: Individual medical costs billed by health insurance

**NOTE :**

**Just upload class1.ipynb in the google colab and just upload the dataset train1 and test1.after that your code is ready to run.**

**CODE :**

***# -\*- coding: utf-8 -\*-***

***"""class1\_advreg.ipynb***

***Automatically generated by Colaboratory.***

***Original file is located at***

***https://colab.research.google.com/drive/1S4-gdtkwEx3C0L8yH4H4ELDuQgZaOVQ8***

***"""***

**import pandas as pd**

**import numpy as np**

**import seaborn as sns**

**data=pd.read\_csv('train1.csv')**

**data1=pd.read\_csv('test1.csv')**

**data1.head()**

**data.head()**

**data.head()**

**type(data)**

**data['MSZoning'].value\_counts()**

**sns.heatmap(data.isnull(),yticklabels=False,cbar=False)**

**data.shape**

**data1.shape**

**train = data.copy()**

**test = data1.copy()**

**train.head()**

**target=train.drop('SalePrice',axis=1)**

**train.shape**

**train.head()**

***# know at this point df,train,data contain same stuff just naming is different***

**df = train**

***#train.drop('SalePrice',axis=0) # for droping column and store to given variable***

**"""apply the feature enginerring"""**

**sns.heatmap(df.isnull(),yticklabels=False,cbar=False)**

**df.isna()**

**df.drop(['Alley'],axis=1,inplace=True)**

**df.drop(['PoolQC','Fence','MiscFeature'],axis=1,inplace=True)**

**sns.heatmap(df.isnull(),yticklabels=False,cbar=False)**

**df['LotFrontage']=df['LotFrontage'].fillna(df['LotFrontage'].mean())**

**df['BsmtCond']=df['BsmtCond'].fillna(df['BsmtCond'].mode()[0])**

**df['BsmtQual']=df['BsmtQual'].fillna(df['BsmtQual'].mode()[0])**

**df['FireplaceQu']=df['FireplaceQu'].fillna(df['FireplaceQu'].mode()[0])**

**df['GarageType']=df['GarageType'].fillna(df['GarageType'].mode()[0])**

**df.drop(['GarageYrBlt'],axis=1,inplace=True)**

**df['GarageFinish']=df['GarageFinish'].fillna(df['GarageFinish'].mode()[0])**

**df['GarageQual']=df['GarageQual'].fillna(df['GarageQual'].mode()[0])**

**df['GarageCond']=df['GarageCond'].fillna(df['GarageCond'].mode()[0])**

**sns.heatmap(df.isnull(),yticklabels=False,cbar=False)**

**df.drop(['Id'],axis=1,inplace=True) *# in this case directly dropping***

**df.isnull().sum()**

**df.isnull().describe()**

**df['MSZoning'].isna()**

**df['MasVnrType']=df['MasVnrType'].fillna(df['MasVnrType'].mode()[0])**

**df['MasVnrArea']=df['MasVnrArea'].fillna(df['MasVnrArea'].mode()[0])**

**df['BsmtExposure']=df['BsmtExposure'].fillna(df['BsmtExposure'].mode()[0])**

**df['BsmtFinType2']=df['BsmtFinType2'].fillna(df['BsmtFinType2'].mode()[0])**

**sns.heatmap(df.isnull(),yticklabels=False,cbar=False)**

**df.isna().any()**

**nan\_cols = [i for i in df.columns if df[i].isnull().any()] *#best way to find which colums having nan values***

**nan\_cols**

**nan\_cols[0]**

**len(nan\_cols)**

**for i in range (len(nan\_cols)):**

**df[nan\_cols[i]]=df[nan\_cols[i]].fillna(df[nan\_cols[i]].mode()[0])**

**sns.heatmap(df.isnull(),yticklabels=False,cbar=False)**

**df.dtypes**

**cat\_features=[i for i in df.columns if df.dtypes[i]=='object'] *#find the categorial features***

**cat\_features**

**len(cat\_features)**

**df.SaleType.value\_counts().sort\_values(ascending=False).head(15)**

**df.SaleType.value\_counts() *#checking the total count***

**final\_df = df.copy()**

**def category\_onehot\_multcols(multcolumns):**

**df\_final=final\_df**

**i=0**

**for fields in multcolumns:**

**print(fields)**

**df1=pd.get\_dummies(final\_df[fields],drop\_first=True)**

**final\_df.drop([fields],axis=1,inplace=True)**

**if i==0:**

**df\_final=df1.copy()**

**else:**

**df\_final=pd.concat([df\_final,df1],axis=1)**

**i=i+1**

**df\_final=pd.concat([final\_df,df\_final],axis=1)**

**return df\_final**

**final\_df=category\_onehot\_multcols(cat\_features)**

**final\_df**

**final\_df =final\_df.loc[:,~final\_df.columns.duplicated()]**

**final\_df.shape**

**df\_Train=final\_df.iloc[:1460,:]**

**df\_Test=final\_df.iloc[1460:,:]**

**df\_Train.shape**

**df\_Train.head()**

***#actual code***

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

**X = df\_Train.drop(['SalePrice'], axis=1)**

**y = df\_Train["SalePrice"]**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.22, random\_state = 5)**

**import xgboost**

**n\_estimators = [100, 500, 900, 1100, 1500]**

**max\_depth = [2, 3, 5, 10, 15]**

**booster = ['gbtree', 'gblinear']**

**base\_score = [0.25, 0.5, 0.75, 1]**

**learning\_rate = [0.05, 0.1, 0.15, 0.20]**

**min\_child\_weight = [1, 2, 3, 4]**

***# Define the grid of hyperparameters to search***

**hyperparameter\_grid = {**

**'n\_estimators': n\_estimators,**

**'max\_depth': max\_depth,**

**'learning\_rate' : learning\_rate,**

**'min\_child\_weight' : min\_child\_weight,**

**'booster' : booster,**

**'base\_score' : base\_score**

**}**

**regressor = xgboost.XGBRegressor(base\_score=0.25,**

**booster='gbtree',**

**learning\_rate=0.1,**

**max\_delta\_step=0,**

**max\_depth=2,**

**min\_child\_weight=1,**

**n\_estimators=900,**

**verbosity=1)**

**regressor.fit(X\_train, y\_train)**

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

**pred = pd.DataFrame(y\_pred)**

**type(pred)**

**type(regressor)**

**type(y\_test)**

***#https://www.kaggle.com/c/titanic/data***

**type(y\_pred)**

**a = y\_test.to\_numpy()**

**type(a)**

***##Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.***

**import sklearn**

**sklearn.metrics.r2\_score(a, y\_pred)**

**RESULT :**

**R2\_score we get 0.9129844018426645 approx 91%**