{

"cells": [

{

"cell\_type": "code",

"execution\_count": 2,

"metadata": {},

"outputs": [],

"source": [

"## CAPT STONE PROJECT - > Credit card Fraud Detection :\n",

"\n",

"# problem statement and Aim of the project : Although digital transactions in India registered a 51% growth in 2018-19, \n",

"# their safety always remains a concern . Fraudalent activities have increased several fold with around 52,304 cases of debit or credit \n",

"# fraud reported in FY'19 alone. Hence application of Machine Learning Techniques can help in detecting such fraudalanet transactions.\n",

"\n",

"\n",

"# Aim of the project : The main objective of the project is to understand which Machine Learning to be used for detecting such fraudalent\n",

"# transactions upfront . In adddition to this we shall also learn about how to deal with class imbalance and steps of Machine Learning Methodology \n",

"# that includes model selection and hyper parameter tuning . Banks needs to be cautious about such customer's transactions as they cannot \n",

"# afford to lose their customer's money to fraudsters . \n",

"\n",

"# The aim of the project is to predict such fraudelent transactions using Machine Learning Algorithm.\n"

]

},

{

"cell\_type": "code",

"execution\_count": 3,

"metadata": {},

"outputs": [],

"source": [

"# Project steps :\n",

"\n",

"# 1) Data Reading & Understanding \n",

"# 2) Data Cleaning and handling of class imbalance \n",

"# 3) Train test split \n",

"# 4) Model Building & hyper parameter tuning \n",

"# 5) Model seelction \n",

"# 6) Model evaluation metrics \n",

"# 7) Conclusion and Recommendation "

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# 1) Data Understanding : Let;s understand the data first :\n",

"\n",

"# importing several libraries :\n",

"\n",

"import numpy as np \n",

"import pandas as pd \n",

"import matplotlib.pyplot as plt\n",

"%matplotlib inline\n",

"import seaborn as sns\n",

"import os \n",

"\n",

"import warnings \n",

"warnings.filterwarnings('ignore')\n",

"\n",

"\n",

"from sklearn import metrics\n",

"from sklearn import preprocessing\n",

"\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"##Step 1: Data Reading & Data Understanding "

]

},

{

"cell\_type": "code",

"execution\_count": 5,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Time</th>\n",

" <th>V1</th>\n",

" <th>V2</th>\n",

" <th>V3</th>\n",

" <th>V4</th>\n",

" <th>V5</th>\n",

" <th>V6</th>\n",

" <th>V7</th>\n",

" <th>V8</th>\n",

" <th>V9</th>\n",

" <th>...</th>\n",

" <th>V21</th>\n",

" <th>V22</th>\n",

" <th>V23</th>\n",

" <th>V24</th>\n",

" <th>V25</th>\n",

" <th>V26</th>\n",

" <th>V27</th>\n",

" <th>V28</th>\n",

" <th>Amount</th>\n",

" <th>Class</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <td>0</td>\n",

" <td>0.0</td>\n",

" <td>-1.359807</td>\n",

" <td>-0.072781</td>\n",

" <td>2.536347</td>\n",

" <td>1.378155</td>\n",

" <td>-0.338321</td>\n",

" <td>0.462388</td>\n",

" <td>0.239599</td>\n",

" <td>0.098698</td>\n",

" <td>0.363787</td>\n",

" <td>...</td>\n",

" <td>-0.018307</td>\n",

" <td>0.277838</td>\n",

" <td>-0.110474</td>\n",

" <td>0.066928</td>\n",

" <td>0.128539</td>\n",

" <td>-0.189115</td>\n",

" <td>0.133558</td>\n",

" <td>-0.021053</td>\n",

" <td>149.62</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <td>1</td>\n",

" <td>0.0</td>\n",

" <td>1.191857</td>\n",

" <td>0.266151</td>\n",

" <td>0.166480</td>\n",

" <td>0.448154</td>\n",

" <td>0.060018</td>\n",

" <td>-0.082361</td>\n",

" <td>-0.078803</td>\n",

" <td>0.085102</td>\n",

" <td>-0.255425</td>\n",

" <td>...</td>\n",

" <td>-0.225775</td>\n",

" <td>-0.638672</td>\n",

" <td>0.101288</td>\n",

" <td>-0.339846</td>\n",

" <td>0.167170</td>\n",

" <td>0.125895</td>\n",

" <td>-0.008983</td>\n",

" <td>0.014724</td>\n",

" <td>2.69</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <td>2</td>\n",

" <td>1.0</td>\n",

" <td>-1.358354</td>\n",

" <td>-1.340163</td>\n",

" <td>1.773209</td>\n",

" <td>0.379780</td>\n",

" <td>-0.503198</td>\n",

" <td>1.800499</td>\n",

" <td>0.791461</td>\n",

" <td>0.247676</td>\n",

" <td>-1.514654</td>\n",

" <td>...</td>\n",

" <td>0.247998</td>\n",

" <td>0.771679</td>\n",

" <td>0.909412</td>\n",

" <td>-0.689281</td>\n",

" <td>-0.327642</td>\n",

" <td>-0.139097</td>\n",

" <td>-0.055353</td>\n",

" <td>-0.059752</td>\n",

" <td>378.66</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <td>3</td>\n",

" <td>1.0</td>\n",

" <td>-0.966272</td>\n",

" <td>-0.185226</td>\n",

" <td>1.792993</td>\n",

" <td>-0.863291</td>\n",

" <td>-0.010309</td>\n",

" <td>1.247203</td>\n",

" <td>0.237609</td>\n",

" <td>0.377436</td>\n",

" <td>-1.387024</td>\n",

" <td>...</td>\n",

" <td>-0.108300</td>\n",

" <td>0.005274</td>\n",

" <td>-0.190321</td>\n",

" <td>-1.175575</td>\n",

" <td>0.647376</td>\n",

" <td>-0.221929</td>\n",

" <td>0.062723</td>\n",

" <td>0.061458</td>\n",

" <td>123.50</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <td>4</td>\n",

" <td>2.0</td>\n",

" <td>-1.158233</td>\n",

" <td>0.877737</td>\n",

" <td>1.548718</td>\n",

" <td>0.403034</td>\n",

" <td>-0.407193</td>\n",

" <td>0.095921</td>\n",

" <td>0.592941</td>\n",

" <td>-0.270533</td>\n",

" <td>0.817739</td>\n",

" <td>...</td>\n",

" <td>-0.009431</td>\n",

" <td>0.798278</td>\n",

" <td>-0.137458</td>\n",

" <td>0.141267</td>\n",

" <td>-0.206010</td>\n",

" <td>0.502292</td>\n",

" <td>0.219422</td>\n",

" <td>0.215153</td>\n",

" <td>69.99</td>\n",

" <td>0</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"<p>5 rows Ã— 31 columns</p>\n",

"</div>"

],

"text/plain": [

" Time V1 V2 V3 V4 V5 V6 V7 \\\n",

"0 0.0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 0.239599 \n",

"1 0.0 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 -0.078803 \n",

"2 1.0 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461 \n",

"3 1.0 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 0.237609 \n",

"4 2.0 -1.158233 0.877737 1.548718 0.403034 -0.407193 0.095921 0.592941 \n",

"\n",

" V8 V9 ... V21 V22 V23 V24 V25 \\\n",

"0 0.098698 0.363787 ... -0.018307 0.277838 -0.110474 0.066928 0.128539 \n",

"1 0.085102 -0.255425 ... -0.225775 -0.638672 0.101288 -0.339846 0.167170 \n",

"2 0.247676 -1.514654 ... 0.247998 0.771679 0.909412 -0.689281 -0.327642 \n",

"3 0.377436 -1.387024 ... -0.108300 0.005274 -0.190321 -1.175575 0.647376 \n",

"4 -0.270533 0.817739 ... -0.009431 0.798278 -0.137458 0.141267 -0.206010 \n",

"\n",

" V26 V27 V28 Amount Class \n",

"0 -0.189115 0.133558 -0.021053 149.62 0 \n",

"1 0.125895 -0.008983 0.014724 2.69 0 \n",

"2 -0.139097 -0.055353 -0.059752 378.66 0 \n",

"3 -0.221929 0.062723 0.061458 123.50 0 \n",

"4 0.502292 0.219422 0.215153 69.99 0 \n",

"\n",

"[5 rows x 31 columns]"

]

},

"execution\_count": 5,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

" \n",

"\n",

"\n",

"df\_credit=pd.read\_csv(r'C:\\Users\\amuly\\Desktop\\CAPSTONE PROJECT\\creditcard1.csv')\n",

"\n",

"df\_credit.head(5)\n"

]

},

{

"cell\_type": "code",

"execution\_count": 6,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"(284807, 31)"

]

},

"execution\_count": 6,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"df\_credit.shape\n",

"\n"

]

},

{

"cell\_type": "code",

"execution\_count": 7,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"<class 'pandas.core.frame.DataFrame'>\n",

"RangeIndex: 284807 entries, 0 to 284806\n",

"Data columns (total 31 columns):\n",

"Time 284807 non-null float64\n",

"V1 284807 non-null float64\n",

"V2 284807 non-null float64\n",

"V3 284807 non-null float64\n",

"V4 284807 non-null float64\n",

"V5 284807 non-null float64\n",

"V6 284807 non-null float64\n",

"V7 284807 non-null float64\n",

"V8 284807 non-null float64\n",

"V9 284807 non-null float64\n",

"V10 284807 non-null float64\n",

"V11 284807 non-null float64\n",

"V12 284807 non-null float64\n",

"V13 284807 non-null float64\n",

"V14 284807 non-null float64\n",

"V15 284807 non-null float64\n",

"V16 284807 non-null float64\n",

"V17 284807 non-null float64\n",

"V18 284807 non-null float64\n",

"V19 284807 non-null float64\n",

"V20 284807 non-null float64\n",

"V21 284807 non-null float64\n",

"V22 284807 non-null float64\n",

"V23 284807 non-null float64\n",

"V24 284807 non-null float64\n",

"V25 284807 non-null float64\n",

"V26 284807 non-null float64\n",

"V27 284807 non-null float64\n",

"V28 284807 non-null float64\n",

"Amount 284807 non-null float64\n",

"Class 284807 non-null int64\n",

"dtypes: float64(30), int64(1)\n",

"memory usage: 67.4 MB\n"

]

}

],

"source": [

"df\_credit.info()"

]

},

{

"cell\_type": "code",

"execution\_count": 8,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Time</th>\n",

" <th>V1</th>\n",

" <th>V2</th>\n",

" <th>V3</th>\n",

" <th>V4</th>\n",

" <th>V5</th>\n",

" <th>V6</th>\n",

" <th>V7</th>\n",

" <th>V8</th>\n",

" <th>V9</th>\n",

" <th>...</th>\n",

" <th>V21</th>\n",

" <th>V22</th>\n",

" <th>V23</th>\n",

" <th>V24</th>\n",

" <th>V25</th>\n",

" <th>V26</th>\n",

" <th>V27</th>\n",

" <th>V28</th>\n",

" <th>Amount</th>\n",

" <th>Class</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <td>count</td>\n",

" <td>284807.000000</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>...</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>2.848070e+05</td>\n",

" <td>284807.000000</td>\n",

" <td>284807.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <td>mean</td>\n",

" <td>94813.859575</td>\n",

" <td>3.919560e-15</td>\n",

" <td>5.688174e-16</td>\n",

" <td>-8.769071e-15</td>\n",

" <td>2.782312e-15</td>\n",

" <td>-1.552563e-15</td>\n",

" <td>2.010663e-15</td>\n",

" <td>-1.694249e-15</td>\n",

" <td>-1.927028e-16</td>\n",

" <td>-3.137024e-15</td>\n",

" <td>...</td>\n",

" <td>1.537294e-16</td>\n",

" <td>7.959909e-16</td>\n",

" <td>5.367590e-16</td>\n",

" <td>4.458112e-15</td>\n",

" <td>1.453003e-15</td>\n",

" <td>1.699104e-15</td>\n",

" <td>-3.660161e-16</td>\n",

" <td>-1.206049e-16</td>\n",

" <td>88.349619</td>\n",

" <td>0.001727</td>\n",

" </tr>\n",

" <tr>\n",

" <td>std</td>\n",

" <td>47488.145955</td>\n",

" <td>1.958696e+00</td>\n",

" <td>1.651309e+00</td>\n",

" <td>1.516255e+00</td>\n",

" <td>1.415869e+00</td>\n",

" <td>1.380247e+00</td>\n",

" <td>1.332271e+00</td>\n",

" <td>1.237094e+00</td>\n",

" <td>1.194353e+00</td>\n",

" <td>1.098632e+00</td>\n",

" <td>...</td>\n",

" <td>7.345240e-01</td>\n",

" <td>7.257016e-01</td>\n",

" <td>6.244603e-01</td>\n",

" <td>6.056471e-01</td>\n",

" <td>5.212781e-01</td>\n",

" <td>4.822270e-01</td>\n",

" <td>4.036325e-01</td>\n",

" <td>3.300833e-01</td>\n",

" <td>250.120109</td>\n",

" <td>0.041527</td>\n",

" </tr>\n",

" <tr>\n",

" <td>min</td>\n",

" <td>0.000000</td>\n",

" <td>-5.640751e+01</td>\n",

" <td>-7.271573e+01</td>\n",

" <td>-4.832559e+01</td>\n",

" <td>-5.683171e+00</td>\n",

" <td>-1.137433e+02</td>\n",

" <td>-2.616051e+01</td>\n",

" <td>-4.355724e+01</td>\n",

" <td>-7.321672e+01</td>\n",

" <td>-1.343407e+01</td>\n",

" <td>...</td>\n",

" <td>-3.483038e+01</td>\n",

" <td>-1.093314e+01</td>\n",

" <td>-4.480774e+01</td>\n",

" <td>-2.836627e+00</td>\n",

" <td>-1.029540e+01</td>\n",

" <td>-2.604551e+00</td>\n",

" <td>-2.256568e+01</td>\n",

" <td>-1.543008e+01</td>\n",

" <td>0.000000</td>\n",

" <td>0.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <td>25%</td>\n",

" <td>54201.500000</td>\n",

" <td>-9.203734e-01</td>\n",

" <td>-5.985499e-01</td>\n",

" <td>-8.903648e-01</td>\n",

" <td>-8.486401e-01</td>\n",

" <td>-6.915971e-01</td>\n",

" <td>-7.682956e-01</td>\n",

" <td>-5.540759e-01</td>\n",

" <td>-2.086297e-01</td>\n",

" <td>-6.430976e-01</td>\n",

" <td>...</td>\n",

" <td>-2.283949e-01</td>\n",

" <td>-5.423504e-01</td>\n",

" <td>-1.618463e-01</td>\n",

" <td>-3.545861e-01</td>\n",

" <td>-3.171451e-01</td>\n",

" <td>-3.269839e-01</td>\n",

" <td>-7.083953e-02</td>\n",

" <td>-5.295979e-02</td>\n",

" <td>5.600000</td>\n",

" <td>0.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <td>50%</td>\n",

" <td>84692.000000</td>\n",

" <td>1.810880e-02</td>\n",

" <td>6.548556e-02</td>\n",

" <td>1.798463e-01</td>\n",

" <td>-1.984653e-02</td>\n",

" <td>-5.433583e-02</td>\n",

" <td>-2.741871e-01</td>\n",

" <td>4.010308e-02</td>\n",

" <td>2.235804e-02</td>\n",

" <td>-5.142873e-02</td>\n",

" <td>...</td>\n",

" <td>-2.945017e-02</td>\n",

" <td>6.781943e-03</td>\n",

" <td>-1.119293e-02</td>\n",

" <td>4.097606e-02</td>\n",

" <td>1.659350e-02</td>\n",

" <td>-5.213911e-02</td>\n",

" <td>1.342146e-03</td>\n",

" <td>1.124383e-02</td>\n",

" <td>22.000000</td>\n",

" <td>0.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <td>75%</td>\n",

" <td>139320.500000</td>\n",

" <td>1.315642e+00</td>\n",

" <td>8.037239e-01</td>\n",

" <td>1.027196e+00</td>\n",

" <td>7.433413e-01</td>\n",

" <td>6.119264e-01</td>\n",

" <td>3.985649e-01</td>\n",

" <td>5.704361e-01</td>\n",

" <td>3.273459e-01</td>\n",

" <td>5.971390e-01</td>\n",

" <td>...</td>\n",

" <td>1.863772e-01</td>\n",

" <td>5.285536e-01</td>\n",

" <td>1.476421e-01</td>\n",

" <td>4.395266e-01</td>\n",

" <td>3.507156e-01</td>\n",

" <td>2.409522e-01</td>\n",

" <td>9.104512e-02</td>\n",

" <td>7.827995e-02</td>\n",

" <td>77.165000</td>\n",

" <td>0.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <td>max</td>\n",

" <td>172792.000000</td>\n",

" <td>2.454930e+00</td>\n",

" <td>2.205773e+01</td>\n",

" <td>9.382558e+00</td>\n",

" <td>1.687534e+01</td>\n",

" <td>3.480167e+01</td>\n",

" <td>7.330163e+01</td>\n",

" <td>1.205895e+02</td>\n",

" <td>2.000721e+01</td>\n",

" <td>1.559499e+01</td>\n",

" <td>...</td>\n",

" <td>2.720284e+01</td>\n",

" <td>1.050309e+01</td>\n",

" <td>2.252841e+01</td>\n",

" <td>4.584549e+00</td>\n",

" <td>7.519589e+00</td>\n",

" <td>3.517346e+00</td>\n",

" <td>3.161220e+01</td>\n",

" <td>3.384781e+01</td>\n",

" <td>25691.160000</td>\n",

" <td>1.000000</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"<p>8 rows Ã— 31 columns</p>\n",

"</div>"

],

"text/plain": [

" Time V1 V2 V3 V4 \\\n",

"count 284807.000000 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 \n",

"mean 94813.859575 3.919560e-15 5.688174e-16 -8.769071e-15 2.782312e-15 \n",

"std 47488.145955 1.958696e+00 1.651309e+00 1.516255e+00 1.415869e+00 \n",

"min 0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00 \n",

"25% 54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01 \n",

"50% 84692.000000 1.810880e-02 6.548556e-02 1.798463e-01 -1.984653e-02 \n",

"75% 139320.500000 1.315642e+00 8.037239e-01 1.027196e+00 7.433413e-01 \n",

"max 172792.000000 2.454930e+00 2.205773e+01 9.382558e+00 1.687534e+01 \n",

"\n",

" V5 V6 V7 V8 V9 \\\n",

"count 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 \n",

"mean -1.552563e-15 2.010663e-15 -1.694249e-15 -1.927028e-16 -3.137024e-15 \n",

"std 1.380247e+00 1.332271e+00 1.237094e+00 1.194353e+00 1.098632e+00 \n",

"min -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01 \n",

"25% -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01 \n",

"50% -5.433583e-02 -2.741871e-01 4.010308e-02 2.235804e-02 -5.142873e-02 \n",

"75% 6.119264e-01 3.985649e-01 5.704361e-01 3.273459e-01 5.971390e-01 \n",

"max 3.480167e+01 7.330163e+01 1.205895e+02 2.000721e+01 1.559499e+01 \n",

"\n",

" ... V21 V22 V23 V24 \\\n",

"count ... 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 \n",

"mean ... 1.537294e-16 7.959909e-16 5.367590e-16 4.458112e-15 \n",

"std ... 7.345240e-01 7.257016e-01 6.244603e-01 6.056471e-01 \n",

"min ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00 \n",

"25% ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01 \n",

"50% ... -2.945017e-02 6.781943e-03 -1.119293e-02 4.097606e-02 \n",

"75% ... 1.863772e-01 5.285536e-01 1.476421e-01 4.395266e-01 \n",

"max ... 2.720284e+01 1.050309e+01 2.252841e+01 4.584549e+00 \n",

"\n",

" V25 V26 V27 V28 Amount \\\n",

"count 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 284807.000000 \n",

"mean 1.453003e-15 1.699104e-15 -3.660161e-16 -1.206049e-16 88.349619 \n",

"std 5.212781e-01 4.822270e-01 4.036325e-01 3.300833e-01 250.120109 \n",

"min -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01 0.000000 \n",

"25% -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02 5.600000 \n",

"50% 1.659350e-02 -5.213911e-02 1.342146e-03 1.124383e-02 22.000000 \n",

"75% 3.507156e-01 2.409522e-01 9.104512e-02 7.827995e-02 77.165000 \n",

"max 7.519589e+00 3.517346e+00 3.161220e+01 3.384781e+01 25691.160000 \n",

"\n",

" Class \n",

"count 284807.000000 \n",

"mean 0.001727 \n",

"std 0.041527 \n",

"min 0.000000 \n",

"25% 0.000000 \n",

"50% 0.000000 \n",

"75% 0.000000 \n",

"max 1.000000 \n",

"\n",

"[8 rows x 31 columns]"

]

},

"execution\_count": 8,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"df\_credit.describe()"

]

},

{

"cell\_type": "code",

"execution\_count": 9,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"0 284315\n",

"1 492\n",

"Name: Class, dtype: int64"

]

},

"execution\_count": 9,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"# As we can see that the data is higly imbalanced in nature with class 1 as Fraud and class 0 as Non Fraudalent transactions \n",

"\n",

"# inference : As from the above we can see that the PCA was already done on the variables using normalisation as there are noticeable amount of \n",

"# outliers in the column \"Amount\" and also the variables \"Time\" & \"Amount\" are not normally distributed . \n",

"\n",

"df\_credit[\"Class\"].value\_counts()"

]

},

{

"cell\_type": "code",

"execution\_count": 10,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<matplotlib.axes.\_subplots.AxesSubplot at 0x11680860bc8>"

]

},

"execution\_count": 10,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Lets plot the class types and see the distribution of data :\n",

"\n",

"sns.barplot(x=\"Class\",y=\"Amount\",data=df\_credit)\n"

]

},

{

"cell\_type": "code",

"execution\_count": 11,

"metadata": {},

"outputs": [],

"source": [

"# Inference & Data Understanding: 1)The data set consists of 284,807 European Credit card transactions with 492 fraudalanet transactions \n",

"# that occured for two days in sep 2013\n",

"\n",

"# 2) Everything except the time and amount has been reduced by a Principle Component Analysis (PCA) for privacy concerns. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'.\n",

"\n",

"# 3) In order to implement a PCA transformation, features need to be previously scaled. So features V1, V2, ... V28 have been scaled already.\n",

"\n",

"# 4) Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.\n",

"\n",

"# 5) The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning.\n",

"\n",

"# 6) Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.\n",

"\n",

"# 7) We can see that the data is highly imbalanced with the total number of fradulant transactions (\"1\") reported to around only 0.172% \n",

"# of the total transactions & the non fraud (\"0\") being reported as nearly 99% . "

]

},

{

"cell\_type": "code",

"execution\_count": 12,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',\n",

" 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',\n",

" 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',\n",

" 'Class'],\n",

" dtype='object')"

]

},

"execution\_count": 12,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"df\_credit.columns"

]

},

{

"cell\_type": "code",

"execution\_count": 13,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Create a scatter plot to see how the distribution of classes is scattered :\n",

"\n",

"plt.scatter(df\_credit.loc[df\_credit['Class'] == 0]['V1'], df\_credit.loc[df\_credit['Class'] == 0]['V2'], label=\"Class #0\", alpha=0.5, linewidth=0.15)\n",

"plt.scatter(df\_credit.loc[df\_credit['Class'] == 1]['V1'], df\_credit.loc[df\_credit['Class'] == 1]['V2'], label=\"Class #1\", alpha=0.5, linewidth=0.15,c='r')\n",

"plt.show()\n"

]

},

{

"cell\_type": "code",

"execution\_count": 14,

"metadata": {},

"outputs": [],

"source": [

"# inference : \n",

"# 1) We can see that the data set contains nearly 284807 entries with class as 0 and class as 1\n",

"\n",

"# 2) There are no \"Null\" nor \"duplicate\" values in the dataset . \n",

"\n",

"# 3) the dataset is highly skewed towards the class 0 being 99% & 0.172% as class 1.\n",

"\n"

]

},

{

"cell\_type": "code",

"execution\_count": 15,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<matplotlib.axes.\_subplots.AxesSubplot at 0x1168079eb88>"

]

},

"execution\_count": 15,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Lets create a scatter plot to see the distribution of class with Time :\n",

"\n",

"# Create a scatter plot to observe the distribution of classes with time\n",

"sns.scatterplot(x=\"Class\", y=\"Time\", data=df\_credit)"

]

},

{

"cell\_type": "code",

"execution\_count": 16,

"metadata": {},

"outputs": [],

"source": [

"# inference : We can see that most of the fradulaent transactions happen during day time "

]

},

{

"cell\_type": "code",

"execution\_count": 17,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<matplotlib.axes.\_subplots.AxesSubplot at 0x11680800a48>"

]

},

"execution\_count": 17,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Lets create a scatter plot with Amount :\n",

"\n",

"sns.scatterplot(x=\"Class\", y=\"Amount\", data=df\_credit)"

]

},

{

"cell\_type": "code",

"execution\_count": 18,

"metadata": {},

"outputs": [],

"source": [

"# Inference : We can see that the mean of the fradulaent transactions comes to around 77 USD . "

]

},

{

"cell\_type": "code",

"execution\_count": 19,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"count 492.000000\n",

"mean 80746.806911\n",

"std 47835.365138\n",

"min 406.000000\n",

"25% 41241.500000\n",

"50% 75568.500000\n",

"75% 128483.000000\n",

"max 170348.000000\n",

"Name: Time, dtype: float64\n"

]

},

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"##Lets see the distribution of Fradulaent (Class 1) activities over Time and Amount :\n",

"\n",

"\n",

"invalid\_df= df\_credit[df\_credit['Class']==1]\n",

"sns.scatterplot(x=\"Amount\", y=\"Time\", data=invalid\_df)\n",

"\n",

"print(invalid\_df.Time.describe())"

]

},

{

"cell\_type": "code",

"execution\_count": 20,

"metadata": {},

"outputs": [],

"source": [

"# Inference : It is infered that most of the fradulaent activities happen during Day time with the mean of 122 USD "

]

},

{

"cell\_type": "code",

"execution\_count": 21,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"count 284315.000000\n",

"mean 94838.202258\n",

"std 47484.015786\n",

"min 0.000000\n",

"25% 54230.000000\n",

"50% 84711.000000\n",

"75% 139333.000000\n",

"max 172792.000000\n",

"Name: Time, dtype: float64\n"

]

},

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# similarly , lets check for Non fradulaent activities with Time & Amount :\n",

"\n",

"\n",

"valid\_df = df\_credit[df\_credit['Class']==0]\n",

"sns.scatterplot(x=\"Amount\", y=\"Time\", data=valid\_df)\n",

"\n",

"print(valid\_df.Time.describe())\n"

]

},

{

"cell\_type": "code",

"execution\_count": 22,

"metadata": {},

"outputs": [],

"source": [

"# Inference : It is seen that mean of Nnn Fradulaent activities is 88 USD. thus the difference between the fraud and non fraud activities is statistically significant"

]

},

{

"cell\_type": "code",

"execution\_count": 23,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# To understand better lets plot the high value transactions in both classes and see how much dollar value impact it causes :\n",

"\n",

"# Plot of high value transactions($200-$2000)\n",

"bins = np.linspace(200, 2000, 100)\n",

"plt.hist(valid\_df.Amount, bins, alpha=1, normed=True, label='Non-Fraud') #Class 0\n",

"plt.hist(invalid\_df.Amount, bins, alpha=1, normed=True, label='Fraud') # Class 1\n",

"plt.legend(loc='upper right')\n",

"plt.title(\"Amount by percentage of transactions (transactions [\\$200-$2000)\")\n](file://\\$200-$2000)\%22)\n)",

"plt.xlabel(\"Transaction amount (USD)\")\n",

"plt.ylabel(\"Percentage of transactions (%)\")\n",

"plt.show()\n"

]

},

{

"cell\_type": "code",

"execution\_count": 24,

"metadata": {},

"outputs": [],

"source": [

"# Inference : It is seen that fraud transactions happen frequently and it seems to be difficult to differentiate between fraud and non fraud with amoutn value "

]

},

{

"cell\_type": "code",

"execution\_count": 25,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"bins = np.linspace(0, 48, 48) #48 hours\n",

"plt.hist((valid\_df.Time/(60\*60)), bins, alpha=1, normed=True, label='Non-Fraud')\n",

"plt.hist((invalid\_df.Time/(60\*60)), bins, alpha=0.6, normed=True, label='Fraud')\n",

"plt.legend(loc='upper right')\n",

"plt.title(\"Percentage of transactions by hour\")\n",

"plt.xlabel(\"Transaction time from first transaction in the dataset (hours)\")\n",

"plt.ylabel(\"Percentage of transactions (%)\")\n",

"plt.show()\n"

]

},

{

"cell\_type": "code",

"execution\_count": 26,

"metadata": {},

"outputs": [],

"source": [

"# inference :We can see that the number of transactions happen mostly at night . \n",

"# Zero hour indicates the hour where the first transaction has happenend \n",

"# there can be seen that the decrease in the number of transactions from hours 1 to 8 . \n",

"\n",

"# Lets plot a Heat Map to understand this better :"

]

},

{

"cell\_type": "code",

"execution\_count": 27,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 1440x720 with 2 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Correlation Matrix :\n",

"\n",

"plt.figure(figsize = (20,10)) # Size of the figure\n",

"sns.heatmap(df\_credit.corr(),annot = True)\n",

"plt.show()"

]

},

{

"cell\_type": "code",

"execution\_count": 28,

"metadata": {},

"outputs": [],

"source": [

"# Inference : \n",

"\n",

"# 1 )Time does not look like a variable really contributing to the distribution. Also the time as per the data dictionary is the time passed since the first transaction which really does not mean anything. So ignoring time column.\n",

"\n",

"# 2) Amount data though seems unrelated, to Class the domain knowledge says amount could be a valid parameter. \n",

" # So keeping the Amount column and not deleting it explicitly.\n",

"\n",

"# 3) None of the PCA variables have any correlation between them but seem to have some correlation with the class variable"

]

},

{

"cell\_type": "code",

"execution\_count": 29,

"metadata": {},

"outputs": [],

"source": [

"# Drop unnecessary columns\n",

"df\_master = df\_credit\n",

"df\_credit.drop(columns=['Time'], inplace=True)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Step 2 : Data Cleaning and handling of class imbalance :"

]

},

{

"cell\_type": "code",

"execution\_count": 30,

"metadata": {},

"outputs": [],

"source": [

"## Step 2 : Data Cleaning and handling of class imbalance :\n",

"\n",

"# Owing to the fact that the data is highly imbalanced , we need to implement the sampling techniques to handle the imbalanced \n",

"# data before splitting to Train -Test Splits . \n",

"\n",

"# Resampling Techniques for Imbalanced data can be done of three types :\n",

"\n",

"# 1) Under Sampling : This is also called Random Under sampling . Here we shall take random samples from the majority class , Non Fraud \n",

"# transactions to match with fradualent transactions . By this way we shall ignore huge amount of data . \n",

"\n",

"# 2) Over Sampling : This is also called Random Over sampling . we simply add weights or take random fraud cases from the minority class\n",

"# though this shall increase the minority population , but still it will make the model become too complex and exagarate by adding \n",

"# many samples . Thus this menthod is also not advisable to follow \n",

"\n",

"# 3) SMOTE : Synthetic Minority Over sampling technique , also called SMOTE adjust the imbalancesin the data by over sampling the \n",

"# minority class or observations using the nearest neighbours of fraud cases to create new synthetic samples /frauds instead of just copying the minority samples.\n",

"\n",

"\n"

]

},

{

"cell\_type": "code",

"execution\_count": 33,

"metadata": {},

"outputs": [

{

"ename": "KeyError",

"evalue": "\"['Time'] not found in axis\"",

"output\_type": "error",

"traceback": [

"\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",

"\u001b[1;31mKeyError\u001b[0m Traceback (most recent call last)",

"\u001b[1;32m<ipython-input-33-b0a53ee52ab7>\u001b[0m in \u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[0;32m 1\u001b[0m \u001b[1;31m# Drop unnecessary columns\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 2\u001b[0m \u001b[0mdf\_master\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mdf\_credit\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m----> 3\u001b[1;33m \u001b[0mdf\_credit\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mdrop\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mcolumns\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;33m[\u001b[0m\u001b[1;34m'Time'\u001b[0m\u001b[1;33m]\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0minplace\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;32mTrue\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m",

"\u001b[1;32m~\\Anaconda3\\lib\\site-packages\\pandas\\core\\frame.py\u001b[0m in \u001b[0;36mdrop\u001b[1;34m(self, labels, axis, index, columns, level, inplace, errors)\u001b[0m\n\u001b[0;32m 4100\u001b[0m \u001b[0mlevel\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0mlevel\u001b[0m\u001b[1;33m,\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 4101\u001b[0m \u001b[0minplace\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0minplace\u001b[0m\u001b[1;33m,\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m-> 4102\u001b[1;33m \u001b[0merrors\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0merrors\u001b[0m\u001b[1;33m,\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 4103\u001b[0m )\n\u001b[0;32m 4104\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;32m~\\Anaconda3\\lib\\site-packages\\pandas\\core\\generic.py\u001b[0m in \u001b[0;36mdrop\u001b[1;34m(self, labels, axis, index, columns, level, inplace, errors)\u001b[0m\n\u001b[0;32m 3912\u001b[0m \u001b[1;32mfor\u001b[0m \u001b[0maxis\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mlabels\u001b[0m \u001b[1;32min\u001b[0m \u001b[0maxes\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mitems\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 3913\u001b[0m \u001b[1;32mif\u001b[0m \u001b[0mlabels\u001b[0m \u001b[1;32mis\u001b[0m \u001b[1;32mnot\u001b[0m \u001b[1;32mNone\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m-> 3914\u001b[1;33m \u001b[0mobj\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mobj\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0m\_drop\_axis\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mlabels\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0maxis\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mlevel\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0mlevel\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0merrors\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0merrors\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 3915\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 3916\u001b[0m \u001b[1;32mif\u001b[0m \u001b[0minplace\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;32m~\\Anaconda3\\lib\\site-packages\\pandas\\core\\generic.py\u001b[0m in \u001b[0;36m\_drop\_axis\u001b[1;34m(self, labels, axis, level, errors)\u001b[0m\n\u001b[0;32m 3944\u001b[0m \u001b[0mnew\_axis\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0maxis\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mdrop\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mlabels\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mlevel\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0mlevel\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0merrors\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0merrors\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 3945\u001b[0m \u001b[1;32melse\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m-> 3946\u001b[1;33m \u001b[0mnew\_axis\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0maxis\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mdrop\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mlabels\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0merrors\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0merrors\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 3947\u001b[0m \u001b[0mresult\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mself\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mreindex\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;33m\*\*\u001b[0m\u001b[1;33m{\u001b[0m\u001b[0maxis\_name\u001b[0m\u001b[1;33m:\u001b[0m \u001b[0mnew\_axis\u001b[0m\u001b[1;33m}\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 3948\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;32m~\\Anaconda3\\lib\\site-packages\\pandas\\core\\indexes\\base.py\u001b[0m in \u001b[0;36mdrop\u001b[1;34m(self, labels, errors)\u001b[0m\n\u001b[0;32m 5338\u001b[0m \u001b[1;32mif\u001b[0m \u001b[0mmask\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0many\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 5339\u001b[0m \u001b[1;32mif\u001b[0m \u001b[0merrors\u001b[0m \u001b[1;33m!=\u001b[0m \u001b[1;34m\"ignore\"\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m-> 5340\u001b[1;33m \u001b[1;32mraise\u001b[0m \u001b[0mKeyError\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;34m\"{} not found in axis\"\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mformat\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mlabels\u001b[0m\u001b[1;33m[\u001b[0m\u001b[0mmask\u001b[0m\u001b[1;33m]\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 5341\u001b[0m \u001b[0mindexer\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mindexer\u001b[0m\u001b[1;33m[\u001b[0m\u001b[1;33m~\u001b[0m\u001b[0mmask\u001b[0m\u001b[1;33m]\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 5342\u001b[0m \u001b[1;32mreturn\u001b[0m \u001b[0mself\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mdelete\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mindexer\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;31mKeyError\u001b[0m: \"['Time'] not found in axis\""

]

}

],

"source": [

"# Drop unnecessary columns\n",

"df\_master = df\_credit\n",

"df\_credit.drop(columns=['Time'], inplace=True)"

]

},

{

"cell\_type": "code",

"execution\_count": 32,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',\n",

" 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',\n",

" 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Class'],\n",

" dtype='object')"

]

},

"execution\_count": 32,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"df\_credit.columns"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Step 3 :Train Test Split :"

]

},

{

"cell\_type": "code",

"execution\_count": 34,

"metadata": {},

"outputs": [],

"source": [

"\n",

"\n",

"y= df\_credit[\"Class\"]\n",

"X = df\_credit.drop(['Class'], axis=1)"

]

},

{

"cell\_type": "code",

"execution\_count": 35,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>V1</th>\n",

" <th>V2</th>\n",

" <th>V3</th>\n",

" <th>V4</th>\n",

" <th>V5</th>\n",

" <th>V6</th>\n",

" <th>V7</th>\n",

" <th>V8</th>\n",

" <th>V9</th>\n",

" <th>V10</th>\n",

" <th>...</th>\n",

" <th>V20</th>\n",

" <th>V21</th>\n",

" <th>V22</th>\n",

" <th>V23</th>\n",

" <th>V24</th>\n",

" <th>V25</th>\n",

" <th>V26</th>\n",

" <th>V27</th>\n",

" <th>V28</th>\n",

" <th>Amount</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <td>0</td>\n",

" <td>-1.359807</td>\n",

" <td>-0.072781</td>\n",

" <td>2.536347</td>\n",

" <td>1.378155</td>\n",

" <td>-0.338321</td>\n",

" <td>0.462388</td>\n",

" <td>0.239599</td>\n",

" <td>0.098698</td>\n",

" <td>0.363787</td>\n",

" <td>0.090794</td>\n",

" <td>...</td>\n",

" <td>0.251412</td>\n",

" <td>-0.018307</td>\n",

" <td>0.277838</td>\n",

" <td>-0.110474</td>\n",

" <td>0.066928</td>\n",

" <td>0.128539</td>\n",

" <td>-0.189115</td>\n",

" <td>0.133558</td>\n",

" <td>-0.021053</td>\n",

" <td>149.62</td>\n",

" </tr>\n",

" <tr>\n",

" <td>1</td>\n",

" <td>1.191857</td>\n",

" <td>0.266151</td>\n",

" <td>0.166480</td>\n",

" <td>0.448154</td>\n",

" <td>0.060018</td>\n",

" <td>-0.082361</td>\n",

" <td>-0.078803</td>\n",

" <td>0.085102</td>\n",

" <td>-0.255425</td>\n",

" <td>-0.166974</td>\n",

" <td>...</td>\n",

" <td>-0.069083</td>\n",

" <td>-0.225775</td>\n",

" <td>-0.638672</td>\n",

" <td>0.101288</td>\n",

" <td>-0.339846</td>\n",

" <td>0.167170</td>\n",

" <td>0.125895</td>\n",

" <td>-0.008983</td>\n",

" <td>0.014724</td>\n",

" <td>2.69</td>\n",

" </tr>\n",

" <tr>\n",

" <td>2</td>\n",

" <td>-1.358354</td>\n",

" <td>-1.340163</td>\n",

" <td>1.773209</td>\n",

" <td>0.379780</td>\n",

" <td>-0.503198</td>\n",

" <td>1.800499</td>\n",

" <td>0.791461</td>\n",

" <td>0.247676</td>\n",

" <td>-1.514654</td>\n",

" <td>0.207643</td>\n",

" <td>...</td>\n",

" <td>0.524980</td>\n",

" <td>0.247998</td>\n",

" <td>0.771679</td>\n",

" <td>0.909412</td>\n",

" <td>-0.689281</td>\n",

" <td>-0.327642</td>\n",

" <td>-0.139097</td>\n",

" <td>-0.055353</td>\n",

" <td>-0.059752</td>\n",

" <td>378.66</td>\n",

" </tr>\n",

" <tr>\n",

" <td>3</td>\n",

" <td>-0.966272</td>\n",

" <td>-0.185226</td>\n",

" <td>1.792993</td>\n",

" <td>-0.863291</td>\n",

" <td>-0.010309</td>\n",

" <td>1.247203</td>\n",

" <td>0.237609</td>\n",

" <td>0.377436</td>\n",

" <td>-1.387024</td>\n",

" <td>-0.054952</td>\n",

" <td>...</td>\n",

" <td>-0.208038</td>\n",

" <td>-0.108300</td>\n",

" <td>0.005274</td>\n",

" <td>-0.190321</td>\n",

" <td>-1.175575</td>\n",

" <td>0.647376</td>\n",

" <td>-0.221929</td>\n",

" <td>0.062723</td>\n",

" <td>0.061458</td>\n",

" <td>123.50</td>\n",

" </tr>\n",

" <tr>\n",

" <td>4</td>\n",

" <td>-1.158233</td>\n",

" <td>0.877737</td>\n",

" <td>1.548718</td>\n",

" <td>0.403034</td>\n",

" <td>-0.407193</td>\n",

" <td>0.095921</td>\n",

" <td>0.592941</td>\n",

" <td>-0.270533</td>\n",

" <td>0.817739</td>\n",

" <td>0.753074</td>\n",

" <td>...</td>\n",

" <td>0.408542</td>\n",

" <td>-0.009431</td>\n",

" <td>0.798278</td>\n",

" <td>-0.137458</td>\n",

" <td>0.141267</td>\n",

" <td>-0.206010</td>\n",

" <td>0.502292</td>\n",

" <td>0.219422</td>\n",

" <td>0.215153</td>\n",

" <td>69.99</td>\n",

" </tr>\n",

" <tr>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" <td>...</td>\n",

" </tr>\n",

" <tr>\n",

" <td>284802</td>\n",

" <td>-11.881118</td>\n",

" <td>10.071785</td>\n",

" <td>-9.834783</td>\n",

" <td>-2.066656</td>\n",

" <td>-5.364473</td>\n",

" <td>-2.606837</td>\n",

" <td>-4.918215</td>\n",

" <td>7.305334</td>\n",

" <td>1.914428</td>\n",

" <td>4.356170</td>\n",

" <td>...</td>\n",

" <td>1.475829</td>\n",

" <td>0.213454</td>\n",

" <td>0.111864</td>\n",

" <td>1.014480</td>\n",

" <td>-0.509348</td>\n",

" <td>1.436807</td>\n",

" <td>0.250034</td>\n",

" <td>0.943651</td>\n",

" <td>0.823731</td>\n",

" <td>0.77</td>\n",

" </tr>\n",

" <tr>\n",

" <td>284803</td>\n",

" <td>-0.732789</td>\n",

" <td>-0.055080</td>\n",

" <td>2.035030</td>\n",

" <td>-0.738589</td>\n",

" <td>0.868229</td>\n",

" <td>1.058415</td>\n",

" <td>0.024330</td>\n",

" <td>0.294869</td>\n",

" <td>0.584800</td>\n",

" <td>-0.975926</td>\n",

" <td>...</td>\n",

" <td>0.059616</td>\n",

" <td>0.214205</td>\n",

" <td>0.924384</td>\n",

" <td>0.012463</td>\n",

" <td>-1.016226</td>\n",

" <td>-0.606624</td>\n",

" <td>-0.395255</td>\n",

" <td>0.068472</td>\n",

" <td>-0.053527</td>\n",

" <td>24.79</td>\n",

" </tr>\n",

" <tr>\n",

" <td>284804</td>\n",

" <td>1.919565</td>\n",

" <td>-0.301254</td>\n",

" <td>-3.249640</td>\n",

" <td>-0.557828</td>\n",

" <td>2.630515</td>\n",

" <td>3.031260</td>\n",

" <td>-0.296827</td>\n",

" <td>0.708417</td>\n",

" <td>0.432454</td>\n",

" <td>-0.484782</td>\n",

" <td>...</td>\n",

" <td>0.001396</td>\n",

" <td>0.232045</td>\n",

" <td>0.578229</td>\n",

" <td>-0.037501</td>\n",

" <td>0.640134</td>\n",

" <td>0.265745</td>\n",

" <td>-0.087371</td>\n",

" <td>0.004455</td>\n",

" <td>-0.026561</td>\n",

" <td>67.88</td>\n",

" </tr>\n",

" <tr>\n",

" <td>284805</td>\n",

" <td>-0.240440</td>\n",

" <td>0.530483</td>\n",

" <td>0.702510</td>\n",

" <td>0.689799</td>\n",

" <td>-0.377961</td>\n",

" <td>0.623708</td>\n",

" <td>-0.686180</td>\n",

" <td>0.679145</td>\n",

" <td>0.392087</td>\n",

" <td>-0.399126</td>\n",

" <td>...</td>\n",

" <td>0.127434</td>\n",

" <td>0.265245</td>\n",

" <td>0.800049</td>\n",

" <td>-0.163298</td>\n",

" <td>0.123205</td>\n",

" <td>-0.569159</td>\n",

" <td>0.546668</td>\n",

" <td>0.108821</td>\n",

" <td>0.104533</td>\n",

" <td>10.00</td>\n",

" </tr>\n",

" <tr>\n",

" <td>284806</td>\n",

" <td>-0.533413</td>\n",

" <td>-0.189733</td>\n",

" <td>0.703337</td>\n",

" <td>-0.506271</td>\n",

" <td>-0.012546</td>\n",

" <td>-0.649617</td>\n",

" <td>1.577006</td>\n",

" <td>-0.414650</td>\n",

" <td>0.486180</td>\n",

" <td>-0.915427</td>\n",

" <td>...</td>\n",

" <td>0.382948</td>\n",

" <td>0.261057</td>\n",

" <td>0.643078</td>\n",

" <td>0.376777</td>\n",

" <td>0.008797</td>\n",

" <td>-0.473649</td>\n",

" <td>-0.818267</td>\n",

" <td>-0.002415</td>\n",

" <td>0.013649</td>\n",

" <td>217.00</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"<p>284807 rows Ã— 29 columns</p>\n",

"</div>"

],

"text/plain": [

" V1 V2 V3 V4 V5 V6 \\\n",

"0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 \n",

"1 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 \n",

"2 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 \n",

"3 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 \n",

"4 -1.158233 0.877737 1.548718 0.403034 -0.407193 0.095921 \n",

"... ... ... ... ... ... ... \n",

"284802 -11.881118 10.071785 -9.834783 -2.066656 -5.364473 -2.606837 \n",

"284803 -0.732789 -0.055080 2.035030 -0.738589 0.868229 1.058415 \n",

"284804 1.919565 -0.301254 -3.249640 -0.557828 2.630515 3.031260 \n",

"284805 -0.240440 0.530483 0.702510 0.689799 -0.377961 0.623708 \n",

"284806 -0.533413 -0.189733 0.703337 -0.506271 -0.012546 -0.649617 \n",

"\n",

" V7 V8 V9 V10 ... V20 V21 \\\n",

"0 0.239599 0.098698 0.363787 0.090794 ... 0.251412 -0.018307 \n",

"1 -0.078803 0.085102 -0.255425 -0.166974 ... -0.069083 -0.225775 \n",

"2 0.791461 0.247676 -1.514654 0.207643 ... 0.524980 0.247998 \n",

"3 0.237609 0.377436 -1.387024 -0.054952 ... -0.208038 -0.108300 \n",

"4 0.592941 -0.270533 0.817739 0.753074 ... 0.408542 -0.009431 \n",

"... ... ... ... ... ... ... ... \n",

"284802 -4.918215 7.305334 1.914428 4.356170 ... 1.475829 0.213454 \n",

"284803 0.024330 0.294869 0.584800 -0.975926 ... 0.059616 0.214205 \n",

"284804 -0.296827 0.708417 0.432454 -0.484782 ... 0.001396 0.232045 \n",

"284805 -0.686180 0.679145 0.392087 -0.399126 ... 0.127434 0.265245 \n",

"284806 1.577006 -0.414650 0.486180 -0.915427 ... 0.382948 0.261057 \n",

"\n",

" V22 V23 V24 V25 V26 V27 V28 \\\n",

"0 0.277838 -0.110474 0.066928 0.128539 -0.189115 0.133558 -0.021053 \n",

"1 -0.638672 0.101288 -0.339846 0.167170 0.125895 -0.008983 0.014724 \n",

"2 0.771679 0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752 \n",

"3 0.005274 -0.190321 -1.175575 0.647376 -0.221929 0.062723 0.061458 \n",

"4 0.798278 -0.137458 0.141267 -0.206010 0.502292 0.219422 0.215153 \n",

"... ... ... ... ... ... ... ... \n",

"284802 0.111864 1.014480 -0.509348 1.436807 0.250034 0.943651 0.823731 \n",

"284803 0.924384 0.012463 -1.016226 -0.606624 -0.395255 0.068472 -0.053527 \n",

"284804 0.578229 -0.037501 0.640134 0.265745 -0.087371 0.004455 -0.026561 \n",

"284805 0.800049 -0.163298 0.123205 -0.569159 0.546668 0.108821 0.104533 \n",

"284806 0.643078 0.376777 0.008797 -0.473649 -0.818267 -0.002415 0.013649 \n",

"\n",

" Amount \n",

"0 149.62 \n",

"1 2.69 \n",

"2 378.66 \n",

"3 123.50 \n",

"4 69.99 \n",

"... ... \n",

"284802 0.77 \n",

"284803 24.79 \n",

"284804 67.88 \n",

"284805 10.00 \n",

"284806 217.00 \n",

"\n",

"[284807 rows x 29 columns]"

]

},

"execution\_count": 35,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"X"

]

},

{

"cell\_type": "code",

"execution\_count": 36,

"metadata": {},

"outputs": [],

"source": [

"from sklearn import model\_selection\n",

"from sklearn.model\_selection import train\_test\_split\n",

"\n",

"X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, test\_size=0.3, random\_state=100, stratify=y)\n",

"\n",

"\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# Train Test Split Classification \n",

"\n",

"# Inference : The data set can be seggregrated into Train , Test and validation sets as the number of data points is considderably high\n",

"# Also the dataset is highly class imbalanced in nature and hence if we take the Train , Validation and Test datset split we may not \n",

"# be sure to what proportion the fraud and non fraud transactions will be in these three sets and hence it is advised to use the \n",

"#Stratified sampling method of Train , Validation and tested datasets as the number of fraud and non fraud transactions in these three sets \n",

"# appear to be in the same proportion as that of the original dataset . \n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Preserve X\_test & y\_test to evaluate on the test data once you build the model"

]

},

{

"cell\_type": "code",

"execution\_count": 37,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"492\n",

"344\n",

"148\n"

]

}

],

"source": [

"print(np.sum(y))\n",

"print(np.sum(y\_train))\n",

"print(np.sum(y\_test))"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Plotting the distribution of a variable"

]

},

{

"cell\_type": "code",

"execution\_count": 38,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 1440x1440 with 30 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# plot the histogram of a variable from the dataset to see the skewness\n",

"df\_credit.hist(figsize=(20,20))\n",

"plt.show()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"##inference : we can see that the data is highly skewed in different variables and hence we need to preprocess the data using Guassian \n",

"## distribution \n",

"\n",

"# - Apply : preprocessing.PowerTransformer(copy=False) to fit & transform the train & test data"

]

},

{

"cell\_type": "code",

"execution\_count": 39,

"metadata": {},

"outputs": [],

"source": [

"from sklearn.preprocessing import PowerTransformer\n",

"pt = PowerTransformer(copy=False)"

]

},

{

"cell\_type": "code",

"execution\_count": 40,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"PowerTransformer(copy=False, method='yeo-johnson', standardize=True)\n"

]

}

],

"source": [

"print(pt.fit(X\_train))"

]

},

{

"cell\_type": "code",

"execution\_count": 41,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"[[ 1.44569155 -0.17985939 -0.82537101 ... -0.21338865 -0.18593313\n",

" -1.44748939]\n",

" [-0.1685569 0.70771555 -0.415595 ... 0.57666235 0.28299091\n",

" -1.13077718]\n",

" [ 1.54055149 0.05920404 -1.39766213 ... -0.23217377 -0.20541154\n",

" -0.86502161]\n",

" ...\n",

" [ 0.44926405 -0.65653239 0.44059444 ... 0.09733236 0.20682708\n",

" 1.18854512]\n",

" [ 0.22075551 -0.98861027 0.05561301 ... -0.18686459 0.25017814\n",

" 1.56343857]\n",

" [-0.45562581 0.76260283 0.24868191 ... -0.11534124 -1.54276014\n",

" -1.31557838]]\n"

]

}

],

"source": [

"print(pt.transform(X\_train))"

]

},

{

"cell\_type": "code",

"execution\_count": 42,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 1440x1440 with 30 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"## Now lets plot the Histogram to see how the data is distributed post transformation of dataset :\n",

"\n",

"X\_train.hist(figsize=(20,20))\n",

"plt.show()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# inference : Post the data transformation is done , we could find the skeweness getting reduced better than before . "

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Step 4 : Model Building on the Imbalanced Data set :\n",

"\n",

"# Lets try to build different classification models on the imbalanced dataset and see the result for further analysis :\n",

"\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#### Step 4 - Model Building -> Logistic Regression :\n",

"\n",

"# Logistic regression is one of the most popular classification models that is easily interpretable and widely used for linearly seperable \n",

"# datset like the ones guven in the problem .Let's build the logistic regression model and see how the model looks like . "

]

},

{

"cell\_type": "code",

"execution\_count": 43,

"metadata": {},

"outputs": [],

"source": [

"# Logistic Regression\n",

"from sklearn import linear\_model #import the package\n",

"from sklearn.linear\_model import LogisticRegression\n",

"from sklearn.feature\_selection import RFE\n",

"from sklearn.model\_selection import StratifiedKFold\n",

"from sklearn.model\_selection import GridSearchCV\n",

"\n",

"import warnings # supress warnings\n",

"warnings.filterwarnings('ignore')\n",

"\n",

"logreg = LogisticRegression()\n",

"\n",

"# specify model\n",

"logreg.fit(X\_train, y\_train)\n",

"\n",

"# creating a StratifiedKFold object with 10 splits \n",

"folds = StratifiedKFold(n\_splits = 10, random\_state = 100)\n"

]

},

{

"cell\_type": "code",

"execution\_count": 44,

"metadata": {},

"outputs": [],

"source": [

"from sklearn.metrics import confusion\_matrix\n",

"from sklearn.metrics import classification\_report, accuracy\_score\n"

]

},

{

"cell\_type": "code",

"execution\_count": 45,

"metadata": {},

"outputs": [],

"source": [

"from sklearn.model\_selection import cross\_val\_score\n",

"from sklearn.model\_selection import cross\_validate"

]

},

{

"cell\_type": "code",

"execution\_count": 46,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([0.90909091, 0.89655172, 0.93333333, 0.875 , 0.86206897,\n",

" 0.89285714, 0.85714286, 0.79166667, 0.92 , 0.92 ])"

]

},

"execution\_count": 46,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"scores = cross\_val\_score(logreg, X\_train, y\_train, scoring='precision', cv=folds)\n",

"scores"

]

},

{

"cell\_type": "code",

"execution\_count": 47,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"{'fit\_time': array([3.21169806, 4.63320088, 4.32062793, 4.5074873 , 3.44983006,\n",

" 3.56196213, 5.22243309, 3.32162452, 3.41430664, 4.91345811]),\n",

" 'score\_time': array([0.359483 , 0.02486634, 0.0167675 , 0.06054354, 0.02623677,\n",

" 0.14042878, 0.02923036, 0.02066469, 0.0282135 , 0.01562309]),\n",

" 'test\_score': array([0.96484775, 0.95446258, 0.99834331, 0.97051409, 0.99391134,\n",

" 0.99833744, 0.97317296, 0.97209119, 0.97220203, 0.97751926])}"

]

},

"execution\_count": 47,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"scores = cross\_validate(logreg, X\_train, y\_train, scoring='roc\_auc', cv=folds)\n",

"scores.keys()\n",

"scores"

]

},

{

"cell\_type": "code",

"execution\_count": 48,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([0.96484775, 0.95446258, 0.99834331, 0.97051409, 0.99391134,\n",

" 0.99833744, 0.97317296, 0.97209119, 0.97220203, 0.97751926])"

]

},

"execution\_count": 48,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"scores = cross\_val\_score(logreg, X\_train, y\_train, scoring='roc\_auc', cv=folds)\n",

"scores"

]

},

{

"cell\_type": "code",

"execution\_count": 49,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"ROC AUC Score: 97.754% (1.398%)\n"

]

}

],

"source": [

"print(\"ROC AUC Score: %.3f%% (%.3f%%)\" % (scores.mean()\*100.0, scores.std()\*100.0))"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# Inference : We can see that the post the cross validation , train and test split the ROC AUC score comes to around 97% which seems to be \n",

"# a good score . But we need to check other model evaluation metrics as well along with the choice of other models to be tesetd \n",

"# to come up with a better model . "

]

},

{

"cell\_type": "code",

"execution\_count": 50,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"[[85248 47]\n",

" [ 68 80]]\n"

]

}

],

"source": [

"y\_pred = logreg.predict(X\_test)\n",

"\n",

"confusion\_matrix = confusion\_matrix(y\_test, y\_pred)\n",

"print(confusion\_matrix)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# The confusion Matrix is one of the good metrics for evaluating any model as in this case , the model is evaluated on the four parameters \n",

"# of sensitivity , Specificity , Precision and Recall parameters . \n",

"\n",

"# we need to aim at predicting the fradulaent transactions as Truely fraud to identify the actual true fraud and recall which is another \n",

"# good metrics that aims to identify the non fradulaent transactions as fraud to be extra cautious . "

]

},

{

"cell\_type": "code",

"execution\_count": 51,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Accuracy of Logistic Regression Model on test set is 0.9986540734758845\n"

]

}

],

"source": [

"# Lets check the Accuracy of the Logistic Regression model :\n",

"\n",

"print('Accuracy of Logistic Regression Model on test set is ',logreg.score(X\_test, y\_test))"

]

},

{

"cell\_type": "code",

"execution\_count": 52,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

" precision recall f1-score support\n",

"\n",

" 0 1.00 1.00 1.00 85295\n",

" 1 0.63 0.54 0.58 148\n",

"\n",

" accuracy 1.00 85443\n",

" macro avg 0.81 0.77 0.79 85443\n",

"weighted avg 1.00 1.00 1.00 85443\n",

"\n"

]

}

],

"source": [

"print(classification\_report(y\_test, y\_pred))"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

" Inference: Though the ROC AUC score on the validation set is great, the confusion matrix on the test set \n",

"shows the performance is not that great for logical regression\n",

"[[85276 19]\n",

" [ 54 94]]"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#### Step 4 - Model Building -> SVM Trial :\n",

"\n",

"from sklearn.svm import SVC\n",

"\n",

"logreg.svm = SVC(kernel='linear') \n",

"logreg.svm.fit(X\_train,y\_train)\n",

"preds = logreg.svm.predict(X\_test)\n",

"metrics.accuracy\_score(y\_test, preds)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"SVM is taking way too long as expected. So not using it for evaluation keeping the time at hand"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#### Step 4 - Model Building -> Decision Tree Classifier :\n",

"\n",

"# Decision Tree model is one of the popular classification models that is easily interpretable even by a lay man and easily built model \n",

"# on considerable large data sets and where the data is linearly interpretable . However we need to be cautious when we use this model \n",

"# as the model has a biggest disadvantage of overfitting the trained set and hence we need to work on hyper parameter tuning \n",

"# that penalises the model by becoming overfit and complex where controls are set on depth of the tree , number of nodes at each branch and number of \n",

"# data points at each node etc . \n"

]

},

{

"cell\_type": "code",

"execution\_count": 53,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=5,\n",

" max\_features=None, max\_leaf\_nodes=None,\n",

" min\_impurity\_decrease=0.0, min\_impurity\_split=None,\n",

" min\_samples\_leaf=1, min\_samples\_split=2,\n",

" min\_weight\_fraction\_leaf=0.0, presort=False,\n",

" random\_state=None, splitter='best')"

]

},

"execution\_count": 53,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"# Importing decision tree classifier from sklearn library\n",

"from sklearn.tree import DecisionTreeClassifier\n",

"# Fitting the decision tree with default hyperparameters, apart from\n",

"# max\_depth which is 5 so that we can plot and read the tree.\n",

"dt\_default = DecisionTreeClassifier(max\_depth=5)\n",

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

]

},

{

"cell\_type": "code",

"execution\_count": 54,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"{'fit\_time': array([9.56953311, 9.3912282 , 9.10637689, 9.51334977, 8.92157507,\n",

" 9.30456638, 9.44977689, 9.03144193, 9.5762732 , 9.15066123]),\n",

" 'score\_time': array([0.04830098, 0.01567984, 0.01562238, 0.01639009, 0.03119946,\n",

" 0.01920819, 0.01566315, 0.01558471, 0.01864624, 0.01562929]),\n",

" 'test\_score': array([0.87147308, 0.89563504, 0.87148241, 0.76768738, 0.95574048,\n",

" 0.8969635 , 0.91167973, 0.77893738, 0.94116169, 0.88212092])}"

]

},

"execution\_count": 54,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"scores = cross\_validate(dt\_default, X\_train, y\_train, scoring='roc\_auc', cv=folds)\n",

"scores.keys()\n",

"scores"

]

},

{

"cell\_type": "code",

"execution\_count": 57,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([0.87147308, 0.89563504, 0.92859942, 0.76768738, 0.92633684,\n",

" 0.86756947, 0.91167973, 0.77893738, 0.94116169, 0.88212092])"

]

},

"execution\_count": 57,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"scores = cross\_val\_score(dt\_default, X\_train, y\_train, scoring='roc\_auc', cv=folds)\n",

"scores"

]

},

{

"cell\_type": "code",

"execution\_count": 58,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"ROC AUC Score: 87.712% (5.699%)\n"

]

}

],

"source": [

"print(\"ROC AUC Score: %.3f%% (%.3f%%)\" % (scores.mean()\*100.0, scores.std()\*100.0))"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"## Inference : decision Tree gives an ROC AUC curve value of 87.712% as compared to Logistic Regression with 97.75% \n"

]

},

{

"cell\_type": "code",

"execution\_count": 59,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

" precision recall f1-score support\n",

"\n",

" 0 1.00 1.00 1.00 85295\n",

" 1 0.69 0.59 0.64 148\n",

"\n",

" accuracy 1.00 85443\n",

" macro avg 0.84 0.79 0.82 85443\n",

"weighted avg 1.00 1.00 1.00 85443\n",

"\n"

]

}

],

"source": [

"# Making predictions\n",

"y\_pred\_default = dt\_default.predict(X\_test)\n",

"\n",

"# Printing classification report\n",

"print(classification\_report(y\_test, y\_pred\_default))"

]

},

{

"cell\_type": "code",

"execution\_count": 60,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([[85256, 39],\n",

" [ 61, 87]], dtype=int64)"

]

},

"execution\_count": 60,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"metrics.confusion\_matrix(y\_test, y\_pred\_default)"

]

},

{

"cell\_type": "code",

"execution\_count": 61,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.9988296291094648\n"

]

}

],

"source": [

"# Printing confusion matrix and accuracy\n",

"print(metrics.accuracy\_score(y\_test,y\_pred\_default))"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"The confusion matrix of DecisionTree is better than the Logical Regresion on the final test data\n",

"[[85275, 20],\n",

"[ 37, 111]]"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#### Step 4 - Model Building -> Random Forest Classifier :\n",

"\n",

"# The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.\n",

"\n",

"#Lets see how this works as one of the model selection algorithm for this problem "

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Random Forest"

]

},

{

"cell\_type": "code",

"execution\_count": 62,

"metadata": {},

"outputs": [],

"source": [

"from sklearn.ensemble import RandomForestClassifier"

]

},

{

"cell\_type": "code",

"execution\_count": 64,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Random Forest Accuracy with Default Hyperparameter 0.9991105181231933\n"

]

}

],

"source": [

"model\_rf = RandomForestClassifier()\n",

"model\_rf.fit(X\_train, y\_train)\n",

"\n",

"# Make predictions\n",

"prediction\_test = model\_rf.predict(X\_test)\n",

"print ('Random Forest Accuracy with Default Hyperparameter',metrics.accuracy\_score(y\_test, prediction\_test))"

]

},

{

"cell\_type": "code",

"execution\_count": 65,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"{'fit\_time': array([32.27503991, 30.3058753 , 34.50752091, 33.76848793, 34.62777996,\n",

" 30.53050208, 35.40348911, 36.4304204 , 36.42460251, 33.82640839]),\n",

" 'score\_time': array([0.06425929, 0.06687284, 0.05086064, 0.16324115, 0.06613183,\n",

" 0.16828322, 0.06228876, 0.08355045, 0.05304646, 0.04854894]),\n",

" 'test\_score': array([0.91386652, 0.92829292, 0.98548531, 0.92830584, 0.95561413,\n",

" 0.95568728, 0.91139229, 0.86689263, 0.94096292, 0.89658518])}"

]

},

"execution\_count": 65,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"scores = cross\_validate(model\_rf, X\_train, y\_train, scoring='roc\_auc', cv=folds)\n",

"scores\n"

]

},

{

"cell\_type": "code",

"execution\_count": 66,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([0.88520465, 0.92831804, 0.97125271, 0.95687368, 0.95562373,\n",

" 0.95569467, 0.91142924, 0.86699312, 0.9408676 , 0.94055209])"

]

},

"execution\_count": 66,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"scores = cross\_val\_score(model\_rf, X\_train, y\_train, scoring='roc\_auc', cv=folds)\n",

"scores"

]

},

{

"cell\_type": "code",

"execution\_count": 67,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"ROC AUC Score: 93.128% (3.206%)\n"

]

}

],

"source": [

"print(\"ROC AUC Score: %.3f%% (%.3f%%)\" % (scores.mean()\*100.0, scores.std()\*100.0))"

]

},

{

"cell\_type": "code",

"execution\_count": 68,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"[[85256 39]\n",

" [ 37 111]]\n"

]

}

],

"source": [

"print(metrics.confusion\_matrix(y\_test, prediction\_test))"

]

},

{

"cell\_type": "code",

"execution\_count": 69,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

" precision recall f1-score support\n",

"\n",

" 0 1.00 1.00 1.00 85295\n",

" 1 0.74 0.75 0.74 148\n",

"\n",

" accuracy 1.00 85443\n",

" macro avg 0.87 0.87 0.87 85443\n",

"weighted avg 1.00 1.00 1.00 85443\n",

"\n"

]

}

],

"source": [

"print(classification\_report(y\_test, prediction\_test))"

]

},

{

"cell\_type": "code",

"execution\_count": 70,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.9991105181231933\n"

]

}

],

"source": [

"print(accuracy\_score(y\_test,prediction\_test))"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"The confusion matrix of the Random Forest looks much better compared to Logical Regression and DecisionTree models\n",

"[[85286 9]\n",

" [ 39 109]]"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"## STEP 4 XG Boost Method :\n",

"#XGBoost stands for eXtreme Gradient Boosting. The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms. \n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"XG Boost"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"from sklearn.ensemble import GradientBoostingClassifier\n",

"from sklearn.tree import DecisionTreeClassifier"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"import xgboost as xgb"

]

},

{

"cell\_type": "code",

"execution\_count": 91,

"metadata": {},

"outputs": [],

"source": [

"shallow\_tree = DecisionTreeClassifier(max\_depth=2, random\_state = 100)"

]

},

{

"cell\_type": "code",

"execution\_count": 92,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"0.9988296291094648"

]

},

"execution\_count": 92,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"# fit the shallow decision tree \n",

"shallow\_tree.fit(X\_train, y\_train)\n",

"\n",

"# test error\n",

"y\_pred = shallow\_tree.predict(X\_test)\n",

"score = metrics.accuracy\_score(y\_test, y\_pred)\n",

"score"

]

},

{

"cell\_type": "code",

"execution\_count": 93,

"metadata": {},

"outputs": [

{

"ename": "NameError",

"evalue": "name 'xgb' is not defined",

"output\_type": "error",

"traceback": [

"\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",

"\u001b[1;31mNameError\u001b[0m Traceback (most recent call last)",

"\u001b[1;32m<ipython-input-93-09921b5ddd72>\u001b[0m in \u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[1;32m----> 1\u001b[1;33m \u001b[0mxgb\_model\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mxgb\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mXGBClassifier\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mobjective\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;34m\"binary:logistic\"\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mrandom\_state\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;36m42\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 2\u001b[0m \u001b[0mxgb\_model\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mfit\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mX\_train\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0my\_train\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 3\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 4\u001b[0m \u001b[0my\_pred\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mxgb\_model\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mpredict\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mX\_test\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;31mNameError\u001b[0m: name 'xgb' is not defined"

]

}

],

"source": [

"xgb\_model = xgb.XGBClassifier(objective=\"binary:logistic\", random\_state=42)\n",

"xgb\_model.fit(X\_train, y\_train)\n",

"\n",

"y\_pred = xgb\_model.predict(X\_test)"

]

},

{

"cell\_type": "code",

"execution\_count": 94,

"metadata": {},

"outputs": [

{

"ename": "NameError",

"evalue": "name 'xgb\_model' is not defined",

"output\_type": "error",

"traceback": [

"\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",

"\u001b[1;31mNameError\u001b[0m Traceback (most recent call last)",

"\u001b[1;32m<ipython-input-94-4b4d5c185825>\u001b[0m in \u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[1;32m----> 1\u001b[1;33m \u001b[0mscores\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mcross\_val\_score\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mxgb\_model\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mX\_train\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0my\_train\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mscoring\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;34m'roc\_auc'\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mcv\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0mfolds\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 2\u001b[0m \u001b[0mscores\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;31mNameError\u001b[0m: name 'xgb\_model' is not defined"

]

}

],

"source": [

"scores = cross\_val\_score(xgb\_model, X\_train, y\_train, scoring='roc\_auc', cv=folds)\n",

"scores"

]

},

{

"cell\_type": "code",

"execution\_count": 95,

"metadata": {},

"outputs": [

{

"ename": "NameError",

"evalue": "name 'xgb\_model' is not defined",

"output\_type": "error",

"traceback": [

"\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",

"\u001b[1;31mNameError\u001b[0m Traceback (most recent call last)",

"\u001b[1;32m<ipython-input-95-179680d132ce>\u001b[0m in \u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[1;32m----> 1\u001b[1;33m \u001b[0mscores\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mcross\_val\_score\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mxgb\_model\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mX\_train\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0my\_train\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mscoring\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;34m'precision'\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mcv\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0mfolds\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 2\u001b[0m \u001b[0mscores\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;31mNameError\u001b[0m: name 'xgb\_model' is not defined"

]

}

],

"source": [

"scores = cross\_val\_score(xgb\_model, X\_train, y\_train, scoring='precision', cv=folds)\n",

"scores"

]

},

{

"cell\_type": "code",

"execution\_count": 96,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"[[85238 57]\n",

" [ 43 105]]\n"

]

}

],

"source": [

"print(metrics.confusion\_matrix(y\_test, y\_pred))"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"## Inference : as we can see that XG Boost has better score of confusion matrix when compared to other models."

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"# Handling Class Imbalance in the dataset:\n",

"\n",

" So far we had built models without working on the class imbalance in the dataset and now lets build the model after applying sampling techniques that handle class imbalance in the dataset "

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"### Print the class distribution after applying SMOTE "

]

},

{

"cell\_type": "code",

"execution\_count": 98,

"metadata": {},

"outputs": [

{

"ename": "ModuleNotFoundError",

"evalue": "No module named 'imblearn'",

"output\_type": "error",

"traceback": [

"\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",

"\u001b[1;31mModuleNotFoundError\u001b[0m Traceback (most recent call last)",

"\u001b[1;32m<ipython-input-98-9dff88493d60>\u001b[0m in \u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[1;32m----> 1\u001b[1;33m \u001b[1;32mfrom\u001b[0m \u001b[0mimblearn\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mover\_sampling\u001b[0m \u001b[1;32mimport\u001b[0m \u001b[0mSMOTE\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mADASYN\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 2\u001b[0m \u001b[1;32mfrom\u001b[0m \u001b[0mimblearn\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mover\_sampling\u001b[0m \u001b[1;32mimport\u001b[0m \u001b[0mRandomOverSampler\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 3\u001b[0m \u001b[1;32mimport\u001b[0m \u001b[0mimblearn\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mover\_sampling\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;31mModuleNotFoundError\u001b[0m: No module named 'imblearn'"

]

}

],

"source": [

"from imblearn.over\_sampling import SMOTE, ADASYN\n",

"from imblearn.over\_sampling import RandomOverSampler\n",

"import imblearn.over\_sampling"

]

},

{

"cell\_type": "code",

"execution\_count": 84,

"metadata": {},

"outputs": [

{

"ename": "NameError",

"evalue": "name 'SMOTE' is not defined",

"output\_type": "error",

"traceback": [

"\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",

"\u001b[1;31mNameError\u001b[0m Traceback (most recent call last)",

"\u001b[1;32m<ipython-input-84-3f0acf86cae2>\u001b[0m in \u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[0;32m 4\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 5\u001b[0m \u001b[1;31m#sm = over\_sampling.SMOTE(random\_state=0)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m----> 6\u001b[1;33m \u001b[0msm\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mSMOTE\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mrandom\_state\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;36m0\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 7\u001b[0m \u001b[0mX\_train\_smote\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0my\_train\_smote\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0msm\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mfit\_resample\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mX\_train\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0my\_train\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 8\u001b[0m \u001b[1;31m# Artificial minority samples and corresponding minority labels from SMOTE are appended\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;31mNameError\u001b[0m: name 'SMOTE' is not defined"

]

}

],

"source": [

"import warnings\n",

"warnings.filterwarnings(\"ignore\")\n",

"\n",

"\n",

"#sm = over\_sampling.SMOTE(random\_state=0)\n",

"sm = SMOTE(random\_state=0)\n",

"X\_train\_smote, y\_train\_smote = sm.fit\_resample(X\_train, y\_train)\n",

"# Artificial minority samples and corresponding minority labels from SMOTE are appended\n",

"# below X\_train and y\_train respectively\n",

"# So to exclusively get the artificial minority samples from SMOTE, we do\n",

"X\_train\_smote\_1 = X\_train\_smote[X\_train.shape[0]:]\n",

"\n",

"X\_train\_1 = X\_train.to\_numpy()[np.where(y\_train==1.0)]\n",

"X\_train\_0 = X\_train.to\_numpy()[np.where(y\_train==0.0)]\n",

"\n",

"\n",

"plt.rcParams['figure.figsize'] = [20, 20]\n",

"fig = plt.figure()\n",

"\n",

"plt.subplot(3, 1, 1)\n",

"plt.scatter(X\_train\_1[:, 0], X\_train\_1[:, 1], label='Actual Class-1 Examples')\n",

"plt.legend()\n",

"\n",

"plt.subplot(3, 1, 2)\n",

"plt.scatter(X\_train\_1[:, 0], X\_train\_1[:, 1], label='Actual Class-1 Examples')\n",

"plt.scatter(X\_train\_smote\_1[:X\_train\_1.shape[0], 0], X\_train\_smote\_1[:X\_train\_1.shape[0], 1],\n",

" label='Artificial SMOTE Class-1 Examples')\n",

"plt.legend()\n",

"\n",

"plt.subplot(3, 1, 3)\n",

"plt.scatter(X\_train\_1[:, 0], X\_train\_1[:, 1], label='Actual Class-1 Examples')\n",

"plt.scatter(X\_train\_0[:X\_train\_1.shape[0], 0], X\_train\_0[:X\_train\_1.shape[0], 1], label='Actual Class-0 Examples')\n",

"plt.legend()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Print the class distribution after applying ADASYN"

]

},

{

"cell\_type": "code",

"execution\_count": 85,

"metadata": {},

"outputs": [

{

"ename": "ModuleNotFoundError",

"evalue": "No module named 'imblearn'",

"output\_type": "error",

"traceback": [

"\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",

"\u001b[1;31mModuleNotFoundError\u001b[0m Traceback (most recent call last)",

"\u001b[1;32m<ipython-input-85-e4669facf626>\u001b[0m in \u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[0;32m 2\u001b[0m \u001b[0mwarnings\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mfilterwarnings\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;34m\"ignore\"\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 3\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m----> 4\u001b[1;33m \u001b[1;32mfrom\u001b[0m \u001b[0mimblearn\u001b[0m \u001b[1;32mimport\u001b[0m \u001b[0mover\_sampling\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 5\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 6\u001b[0m \u001b[0mada\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mover\_sampling\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mADASYN\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mrandom\_state\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;36m0\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;31mModuleNotFoundError\u001b[0m: No module named 'imblearn'"

]

}

],

"source": [

"import warnings\n",

"warnings.filterwarnings(\"ignore\")\n",

"\n",

"from imblearn import over\_sampling\n",

"\n",

"ada = over\_sampling.ADASYN(random\_state=0)\n",

"X\_train\_adasyn, y\_train\_adasyn = ada.fit\_resample(X\_train, y\_train)\n",

"# Artificial minority samples and corresponding minority labels from ADASYN are appended\n",

"# below X\_train and y\_train respectively\n",

"# So to exclusively get the artificial minority samples from ADASYN, we do\n",

"X\_train\_adasyn\_1 = X\_train\_adasyn[X\_train.shape[0]:]\n",

"\n",

"X\_train\_1 = X\_train.to\_numpy()[np.where(y\_train==1.0)]\n",

"X\_train\_0 = X\_train.to\_numpy()[np.where(y\_train==0.0)]\n",

"\n",

"\n",

"\n",

"import matplotlib.pyplot as plt\n",

"%matplotlib inline\n",

"plt.rcParams['figure.figsize'] = [20, 20]\n",

"fig = plt.figure()\n",

"\n",

"plt.subplot(3, 1, 1)\n",

"plt.scatter(X\_train\_1[:, 0], X\_train\_1[:, 1], label='Actual Class-1 Examples')\n",

"plt.legend()\n",

"\n",

"plt.subplot(3, 1, 2)\n",

"plt.scatter(X\_train\_1[:, 0], X\_train\_1[:, 1], label='Actual Class-1 Examples')\n",

"plt.scatter(X\_train\_adasyn\_1[:X\_train\_1.shape[0], 0], X\_train\_adasyn\_1[:X\_train\_1.shape[0], 1],\n",

" label='Artificial ADASYN Class-1 Examples')\n",

"plt.legend()\n",

"\n",

"plt.subplot(3, 1, 3)\n",

"plt.scatter(X\_train\_1[:, 0], X\_train\_1[:, 1], label='Actual Class-1 Examples')\n",

"plt.scatter(X\_train\_0[:X\_train\_1.shape[0], 0], X\_train\_0[:X\_train\_1.shape[0], 1], label='Actual Class-0 Examples')\n",

"plt.legend()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

}

],

"metadata": {

"kernelspec": {

"display\_name": "Python 3",

"language": "python",

"name": "python3"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.7.4"

}

},

"nbformat": 4,

"nbformat\_minor": 2

}