Business Problem Understanding

Dream Housing Finance company deals in all kinds of home loans. They have presence across all urban, semi urban and rural areas. Customer first applies for home loan and after that company validates the customer eligibility for loan.

Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have provided a dataset to identify the customers segments that are eligible for loan amount so that they can specifically target these customers.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv('LoanData.csv')
data.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [4]:
```
# lets check the column names present in the dataset
data.columns
```

Out[4]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```


Data Understanding

- Loan_ID : Unique Loan ID
- Gender : Male/ Female
- Married : Applicant married
- Dependents : Number of dependents
- Education : Applicant Education
- Self_Employed : whether the applicant is Self employed
- ApplicantIncome : Applicant income
- CoapplicantIncome : Coapplicant income
- LoanAmount : Loan amount in thousands
- Loan_Amount_Term : Term of loan in months
- Credit_History : credit history meets guidelines
- Property_Area : Urban/ Semi Urban/ Rural
- Loan_Status : Loan approved **target variable**

In [5]:
```
data['Loan_ID'].nunique()
```

Out[5]:
```
614
```

In [6]:
```
data.drop(columns=['Loan_ID'],inplace=True)
```

In [7]:
```
data['Gender'].unique()
```

Out[7]:
```
array(['Male', 'Female', nan], dtype=object)
```

In [8]:
```
data['Gender'].value_counts()
```

Out[8]:
```
Gender
Male      489
Female    112
Name: count, dtype: int64
```

In [9]:
```
data['Married'].unique()
```

Out[9]:
```
array(['No', 'Yes', nan], dtype=object)
```

In [10]:
```
data["Married"].value_counts()
```

Out[10]:
```
Married
Yes    398
No     213
Name: count, dtype: int64
```

In [11]:
```
data['Dependents'].unique()
```

Out[11]:
```
array(['0', '1', '2', '3+', nan], dtype=object)
```

In [12]:
```
data['Dependents'].value_counts()
```

Out[12]:
```
Dependents
0     345
1     102
2     101
3+     51
Name: count, dtype: int64
```

In [13]:
```
data['Education'].unique()
```

Out[13]:
```
array(['Graduate', 'Not Graduate'], dtype=object)
```

In [14]:
```
data['Education'].value_counts()
```

Out[14]:
```
Education
Graduate        480
Not Graduate    134
Name: count, dtype: int64
```

In [15]:
```
data['Self_Employed'].unique()
```

Out[15]:
```
array(['No', 'Yes', nan], dtype=object)
```

In [16]:
```
data['Self_Employed'].value_counts()
```

Out[16]:
```
Self_Employed
No     500
Yes     82
Name: count, dtype: int64
```

In [17]:
```
data['ApplicantIncome'].unique()
```

Out[17]:
```
array([ 5849,  4583,  3000,  2583,  6000,  5417,  2333,  3036,  4006,
       12841,  3200,  2500,  3073,  1853,  1299,  4950,  3596,  3510,
        4887,  2600,  7660,  5955,  3365,  3717,  9560,  2799,  4226,
        1442,  3750,  4166,  3167,  4692,  3500, 12500,  2275,  1828,
        3667,  3748,  3600,  1800,  2400,  3941,  4695,  3410,  5649,
        5821,  2645,  4000,  1928,  3086,  4230,  4616, 11500,  2708,
        2132,  3366,  8080,  3357,  3029,  2609,  4945,  5726, 10750,
        7100,  4300,  3208,  1875,  4755,  5266,  1000,  3333,  3846,
        2395,  1378,  3988,  2366,  8566,  5695,  2958,  6250,  3273,
        4133,  3620,  6782,  2484,  1977,  4188,  1759,  4288,  4843,
       13650,  4652,  3816,  3052, 11417,  7333,  3800,  2071,  5316,
        2929,  3572,  7451,  5050, 14583,  2214,  5568, 10408,  5667,
        2137,  2957,  3692, 23803,  3865, 10513,  6080, 20166,  2014,
        2718,  3459,  4895,  3316, 14999,  4200,  5042,  6950,  2698,
       11757,  2330, 14866,  1538, 10000,  4860,  6277,  2577,  9166,
        2281,  3254, 39999,  9538,  2980,  1863,  7933,  3089,  4167,
        9323,  3707,  2439,  2237,  8000,  1820, 51763,  3522,  5708,
        4344,  3497,  2045,  5516,  6400,  1916,  4600, 33846,  3625,
       39147,  2178,  2383,   674,  9328,  4885, 12000,  6033,  3858,
        4191,  3125,  8333,  1907,  3416, 11000,  4923,  3992,  3917,
        4408,  3244,  3975,  2479,  3418,  3430,  7787,  5703,  3173,
        3850,   150,  3727,  5000,  4283,  2221,  4009,  2971,  7578,
        3250,  4735,  4758,  2491,  3716,  3189,  3155,  5500,  5746,
        3463,  3812,  3315,  5819,  2510,  2965,  3406,  6050,  9703,
        6608,  2882,  1809,  1668,  3427,  2661, 16250,  3083,  6045,
        5250, 14683,  4931,  6083,  2060,  3481,  7200,  5166,  4095,
        4708,  4333,  2876,  3237, 11146,  2833,  2620,  3900,  2750,
        3993,  3103,  4100,  4053,  3927,  2301,  1811, 20667,  3158,
        3704,  4124,  9508,  3075,  4400,  3153,  4416,  6875,  4666,
        2875,  1625,  2000,  3762, 20233,  7667,  2917,  2927,  2507,
        2473,  3399,  2058,  3541,  4342,  3601,  3166, 15000,  8666,
        4917,  5818,  4384,  2935, 63337,  9833,  5503,  1830,  4160,
        2647,  2378,  4554,  2499,  3523,  6333,  2625,  9083,  8750,
        2666,  2423,  3813,  3875,  5167,  4723,  4750,  3013,  6822,
        6216,  5124,  6325, 19730, 15759,  5185,  3062,  2764,  4817,
        4310,  3069,  5391,  5941,  7167,  4566,  2346,  3010,  5488,
        9167,  9504,  1993,  3100,  3276,  3180,  3033,  3902,  1500,
        2889,  2755,  1963,  7441,  4547,  2167,  2213,  8300, 81000,
        3867,  6256,  6096,  2253,  2149,  2995,  1600,  1025,  3246,
        5829,  2720,  7250, 14880,  4606,  5935,  2920,  2717,  8624,
        6500, 12876,  2425, 10047,  1926, 10416,  7142,  3660,  7901,
        4707, 37719,  3466,  3539,  3340,  2769,  2309,  1958,  3948,
        2483,  7085,  3859,  4301,  3708,  4354,  8334,  2083,  7740,
        3015,  5191,  2947, 16692,   210,  3450,  2653,  4691,  5532,
       16525,  6700,  2873, 16667,  4350,  3095, 10833,  3547, 18333,
        2435,  2699,  5333,  3691, 17263,  3597,  3326,  4625,  2895,
        6283,   645,  3159,  4865,  4050,  3814, 20833,  3583, 13262,
        3598,  6065,  3283,  2130,  5815,  2031,  3074,  4683,  3400,
        2192,  5677,  7948,  4680, 17500,  3775,  5285,  2679,  6783,
        4281,  3588, 11250, 18165,  2550,  6133,  3617,  6417,  4608,
        2138,  3652,  2239,  3017,  2768,  3358,  2526,  2785,  6633,
        2492,  2454,  3593,  5468,  2667, 10139,  3887,  4180,  3675,
       19484,  5923,  5800,  8799,  4467,  3417,  5116, 16666,  6125,
        6406,  3087,  3229,  1782,  3182,  6540,  1836,  1880,  2787,
        2297,  2165,  2726,  9357, 16120,  3833,  6383,  2987,  9963,
        5780,   416,  2894,  3676,  3987,  3232,  2900,  4106,  8072,
        7583], dtype=int64)
```

In [18]:
```
data['CoapplicantIncome'].unique()
```

Out[18]:
```
array([0.00000000e+00, 1.50800000e+03, 2.35800000e+03, 4.19600000e+03,
       1.51600000e+03, 2.50400000e+03, 1.52600000e+03, 1.09680000e+04,
       7.00000000e+02, 1.84000000e+03, 8.10600000e+03, 2.84000000e+03,
       1.08600000e+03, 3.50000000e+03, 5.62500000e+03, 1.91100000e+03,
       1.91700000e+03, 2.92500000e+03, 2.25300000e+03, 1.04000000e+03,
       2.08300000e+03, 3.36900000e+03, 1.66700000e+03, 3.00000000e+03,
       2.06700000e+03, 1.33000000e+03, 1.45900000e+03, 7.21000000e+03,
       1.66800000e+03, 1.21300000e+03, 2.33600000e+03, 3.44000000e+03,
       2.27500000e+03, 1.64400000e+03, 1.16700000e+03, 1.59100000e+03,
       2.20000000e+03, 2.25000000e+03, 2.85900000e+03, 3.79600000e+03,
       3.44900000e+03, 4.59500000e+03, 2.25400000e+03, 3.06600000e+03,
       1.87500000e+03, 1.77400000e+03, 4.75000000e+03, 3.02200000e+03,
       4.00000000e+03, 2.16600000e+03, 1.88100000e+03, 2.53100000e+03,
       2.00000000e+03, 2.11800000e+03, 4.16700000e+03, 2.90000000e+03,
       5.65400000e+03, 1.82000000e+03, 2.30200000e+03, 9.97000000e+02,
       3.54100000e+03, 3.26300000e+03, 3.80600000e+03, 3.58300000e+03,
       7.54000000e+02, 1.03000000e+03, 1.12600000e+03, 3.60000000e+03,
       2.33300000e+03, 4.11400000e+03, 2.28300000e+03, 1.39800000e+03,
       2.14200000e+03, 2.66700000e+03, 8.98000000e+03, 2.01400000e+03,
       1.64000000e+03, 3.85000000e+03, 2.56900000e+03, 1.92900000e+03,
       7.75000000e+03, 1.43000000e+03, 2.03400000e+03, 4.48600000e+03,
       1.42500000e+03, 1.66600000e+03, 8.30000000e+02, 3.75000000e+03,
       1.04100000e+03, 1.28000000e+03, 1.44700000e+03, 3.16600000e+03,
       3.33300000e+03, 1.76900000e+03, 7.36000000e+02, 1.96400000e+03,
       1.61900000e+03, 1.13000000e+04, 1.45100000e+03, 7.25000000e+03,
       5.06300000e+03, 2.13800000e+03, 5.29600000e+03, 2.58300000e+03,
       2.36500000e+03, 2.81600000e+03, 2.50000000e+03, 1.08300000e+03,
       1.25000000e+03, 3.02100000e+03, 9.83000000e+02, 1.80000000e+03,
       1.77500000e+03, 2.38300000e+03, 1.71700000e+03, 2.79100000e+03,
       1.01000000e+03, 1.69500000e+03, 2.05400000e+03, 2.59800000e+03,
       1.77900000e+03, 1.26000000e+03, 5.00000000e+03, 1.98300000e+03,
       5.70100000e+03, 1.30000000e+03, 4.41700000e+03, 4.33300000e+03,
       1.84300000e+03, 1.86800000e+03, 3.89000000e+03, 2.16700000e+03,
       7.10100000e+03, 2.10000000e+03, 4.25000000e+03, 2.20900000e+03,
       3.44700000e+03, 1.38700000e+03, 1.81100000e+03, 1.56000000e+03,
       1.85700000e+03, 2.22300000e+03, 1.84200000e+03, 3.27400000e+03,
       2.42600000e+03, 8.00000000e+02, 9.85799988e+02, 3.05300000e+03,
       2.41600000e+03, 3.33400000e+03, 2.54100000e+03, 2.93400000e+03,
       1.75000000e+03, 1.80300000e+03, 1.86300000e+03, 2.40500000e+03,
       2.13400000e+03, 1.89000000e+02, 1.59000000e+03, 2.98500000e+03,
       4.98300000e+03, 2.16000000e+03, 2.45100000e+03, 1.79300000e+03,
       1.83300000e+03, 4.49000000e+03, 6.88000000e+02, 4.60000000e+03,
       1.58700000e+03, 1.22900000e+03, 2.33000000e+03, 2.45800000e+03,
       3.23000000e+03, 2.16800000e+03, 4.58300000e+03, 6.25000000e+03,
       5.05000000e+02, 3.16700000e+03, 3.66700000e+03, 3.03300000e+03,
       5.26600000e+03, 7.87300000e+03, 1.98700000e+03, 9.23000000e+02,
       4.99600000e+03, 4.23200000e+03, 1.60000000e+03, 3.13600000e+03,
       2.41700000e+03, 2.11500000e+03, 1.62500000e+03, 1.40000000e+03,
       4.84000000e+02, 2.00000000e+04, 2.40000000e+03, 2.03300000e+03,
       3.23700000e+03, 2.77300000e+03, 1.41700000e+03, 1.71900000e+03,
       4.30000000e+03, 1.61200008e+01, 2.34000000e+03, 1.85100000e+03,
       1.12500000e+03, 5.06400000e+03, 1.99300000e+03, 8.33300000e+03,
       1.21000000e+03, 1.37600000e+03, 1.71000000e+03, 1.54200000e+03,
       1.25500000e+03, 1.45600000e+03, 1.73300000e+03, 2.46600000e+03,
       4.08300000e+03, 2.18800000e+03, 1.66400000e+03, 2.91700000e+03,
       2.07900000e+03, 1.50000000e+03, 4.64800000e+03, 1.01400000e+03,
       1.87200000e+03, 1.60300000e+03, 3.15000000e+03, 2.43600000e+03,
       2.78500000e+03, 1.13100000e+03, 2.15700000e+03, 9.13000000e+02,
       1.70000000e+03, 2.85700000e+03, 4.41600000e+03, 3.68300000e+03,
       5.62400000e+03, 5.30200000e+03, 1.48300000e+03, 6.66700000e+03,
       3.01300000e+03, 1.28700000e+03, 2.00400000e+03, 2.03500000e+03,
       6.66600000e+03, 3.66600000e+03, 3.42800000e+03, 1.63200000e+03,
       1.91500000e+03, 1.74200000e+03, 1.42400000e+03, 7.16600000e+03,
       2.08700000e+03, 1.30200000e+03, 5.50000000e+03, 2.04200000e+03,
```

```
       3.90600000e+03, 5.36000000e+02, 2.84500000e+03, 2.52400000e+03,
       6.63000000e+02, 1.95000000e+03, 1.78300000e+03, 2.01600000e+03,
       2.37500000e+03, 3.25000000e+03, 4.26600000e+03, 1.03200000e+03,
       2.66900000e+03, 2.30600000e+03, 2.42000000e+02, 2.06400000e+03,
       4.61000000e+02, 2.21000000e+03, 2.73900000e+03, 2.23200000e+03,
       3.38370000e+04, 1.52200000e+03, 3.41600000e+03, 3.30000000e+03,
       1.00000000e+03, 4.16670000e+04, 2.79200000e+03, 4.30100000e+03,
       3.80000000e+03, 1.41100000e+03, 2.40000000e+02])
```

In [19]:
```
data['LoanAmount'].unique()
```

Out[19]:
```
array([ nan, 128.,  66., 120., 141., 267.,  95., 158., 168., 349.,  70.,
       109., 200., 114.,  17., 125., 100.,  76., 133., 115., 104., 315.,
       116., 112., 151., 191., 122., 110.,  35., 201.,  74., 106., 320.,
       144., 184.,  80.,  47.,  75., 134.,  96.,  88.,  44., 286.,  97.,
       135., 180.,  99., 165., 258., 126., 312., 136., 172.,  81., 187.,
       113., 176., 130., 111., 167., 265.,  50., 210., 175., 131., 188.,
        25., 137., 160., 225., 216.,  94., 139., 152., 118., 185., 154.,
        85., 259., 194.,  93., 370., 182., 650., 102., 290.,  84., 242.,
       129.,  30., 244., 600., 255.,  98., 275., 121.,  63., 700.,  87.,
       101., 495.,  67.,  73., 260., 108.,  58.,  48., 164., 170.,  83.,
        90., 166., 124.,  55.,  59., 127., 214., 240.,  72.,  60., 138.,
        42., 280., 140., 155., 123., 279., 192., 304., 330., 150., 207.,
       436.,  78.,  54.,  89., 143., 105., 132., 480.,  56., 159., 300.,
       376., 117.,  71., 490., 173.,  46., 228., 308., 236., 570., 380.,
       296., 156., 103.,  45.,  65.,  53., 360.,  62., 218., 178., 239.,
       405., 148., 190., 149., 153., 162., 230.,  86., 234., 246., 500.,
       186., 119., 107., 209., 208., 243.,  40., 250., 311., 400., 161.,
       196., 324., 157., 145., 181.,  26., 211.,   9., 205.,  36.,  61.,
       146., 292., 142., 350., 496., 253.])
```

In [20]:
```
data['Loan_Amount_Term'].unique()
```

Out[20]:
```
array([360., 120., 240.,  nan, 180.,  60., 300., 480.,  36.,  84.,  12.])
```

In [21]:
```
data['Loan_Amount_Term'].value_counts()
```

Out[21]:
```
Loan_Amount_Term
360.0    512
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: count, dtype: int64
```

In [22]:
```
data['Credit_History'].unique()
```

Out[22]:
```
array([ 1.,  0., nan])
```

In [23]:
```
data['Credit_History'] = data['Credit_History'].replace({1:"good",0:"bad"})
```

In [24]:
```
data['Credit_History'].unique()
```

Out[24]:
```
array(['good', 'bad', nan], dtype=object)
```

In [25]:
```
data['Credit_History'].value_counts()
```

Out[25]:
```
Credit_History
good    475
bad      89
Name: count, dtype: int64
```

In [26]:
```
data['Property_Area'].unique()
```

Out[26]:
```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

In [27]:
```
data['Property_Area'].value_counts()
```

Out[27]:
```
Property_Area
Semiurban    233
Urban        202
Rural        179
Name: count, dtype: int64
```

In [28]:
```
data['Loan_Status'].unique()
```

Out[28]:
```
array(['Y', 'N'], dtype=object)
```

In [29]:
```
data['Loan_Status'].value_counts()
```

Out[29]:
```
Loan_Status
Y    422
N    192
Name: count, dtype: int64
```

In [30]:
```
continous = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']

discrete_categorical = ['Gender', 'Married', 'Education', 'Self_Employed',
                        'Credit_History', 'Property_Area', 'Loan_Status']

discrete_count = ['Dependents','Loan_Amount_Term']
```

Exploratory Data Analysis (EDA)

for continous Variables

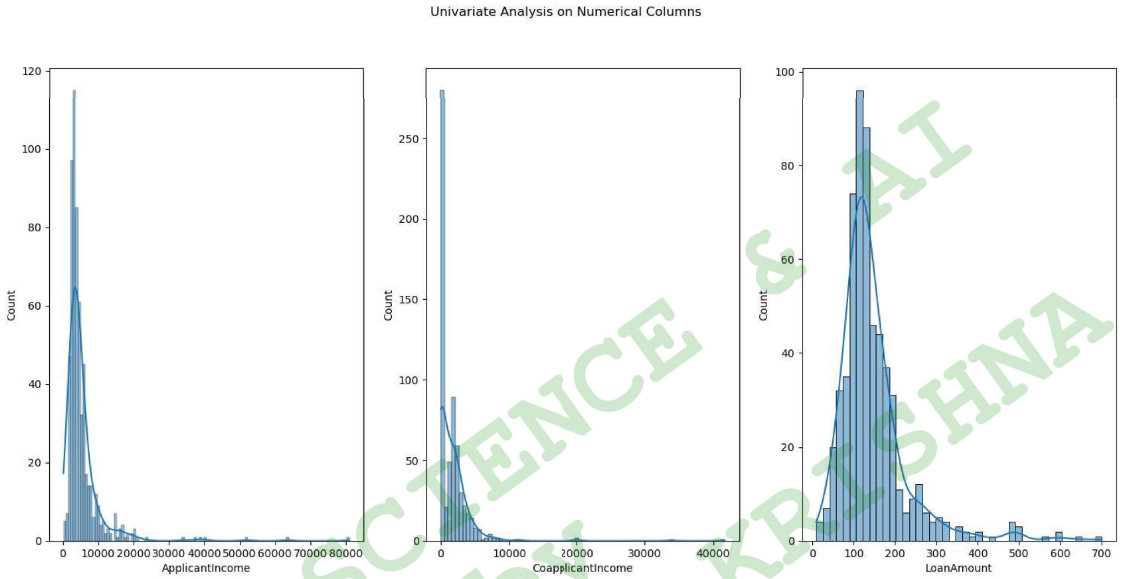
In [31]:
```
data[continous].describe()
```

Out[31]:

	ApplicantIncome	CoapplicantIncome	LoanAmount
count	614.000000	614.000000	592.000000
mean	5403.459283	1621.245798	146.412162
std	6109.041673	2926.248369	85.587325
min	150.000000	0.000000	9.000000
25%	2877.500000	0.000000	100.000000
50%	3812.500000	1188.500000	128.000000
75%	5795.000000	2297.250000	168.000000
max	81000.000000	41667.000000	700.000000

In [32]:
```
plt.rcParams['figure.figsize'] = (18,8)

plt.subplot(1,3, 1)
sns.histplot(data['ApplicantIncome'],kde=True)
```

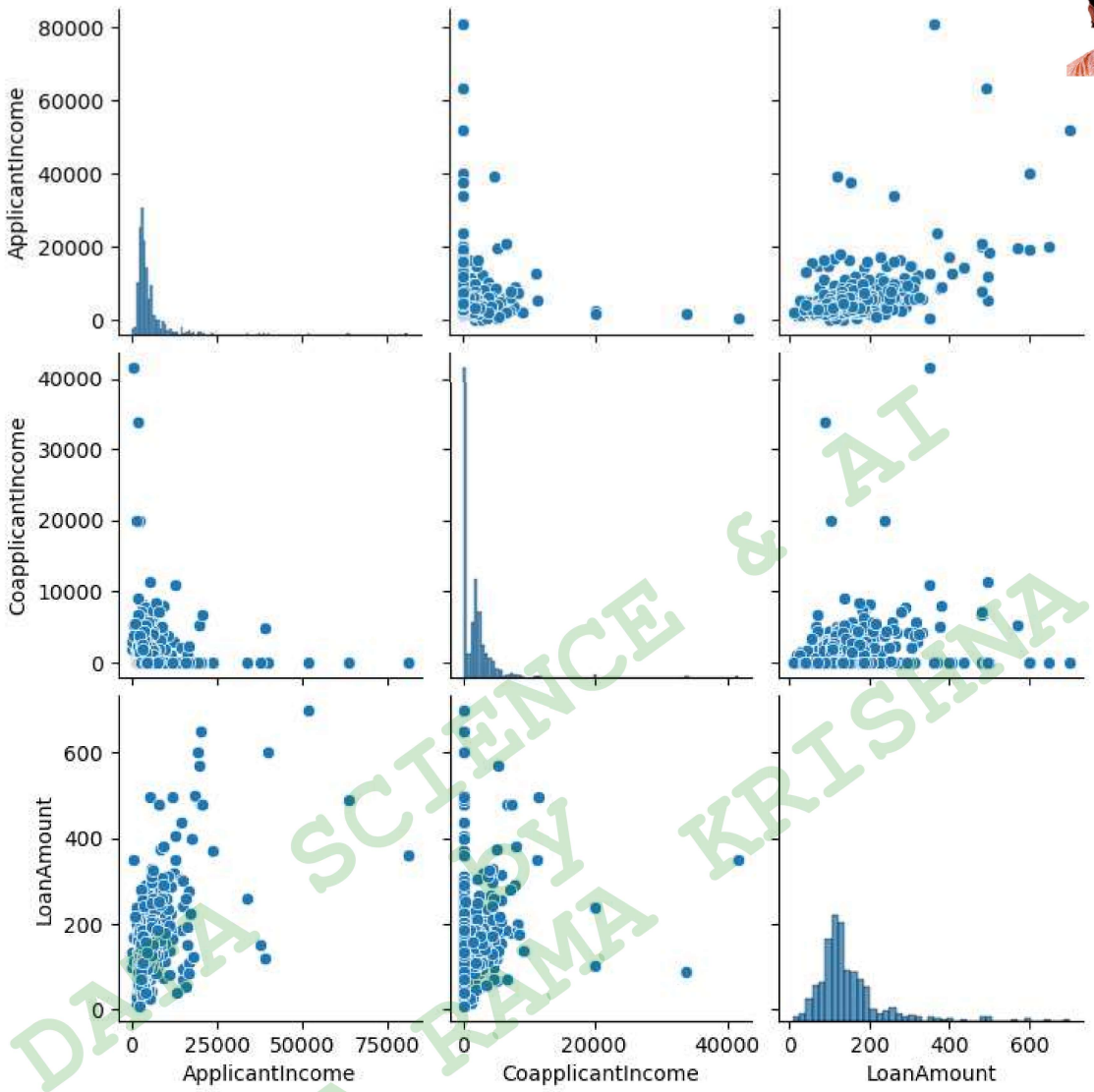
```
plt.subplot(1,3, 2)
sns.histplot(data['CoapplicantIncome'],kde=True)

plt.subplot(1,3, 3)
sns.histplot(data['LoanAmount'], kde=True)

plt.suptitle('Univariate Analysis on Numerical Columns')
plt.show()
```



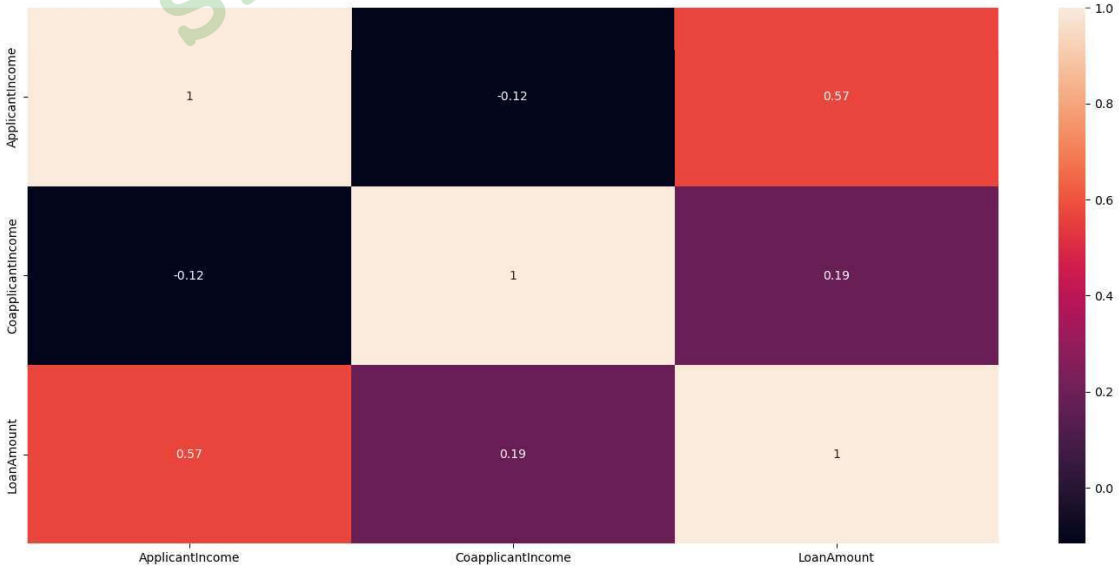
In [33]:
```
data[continous].skew()
```

Out[33]:
```
ApplicantIncome      6.539513
CoapplicantIncome    7.491531
LoanAmount           2.677552
dtype: float64
```

In [34]:
```
sns.pairplot(data[continous])
plt.show()
```

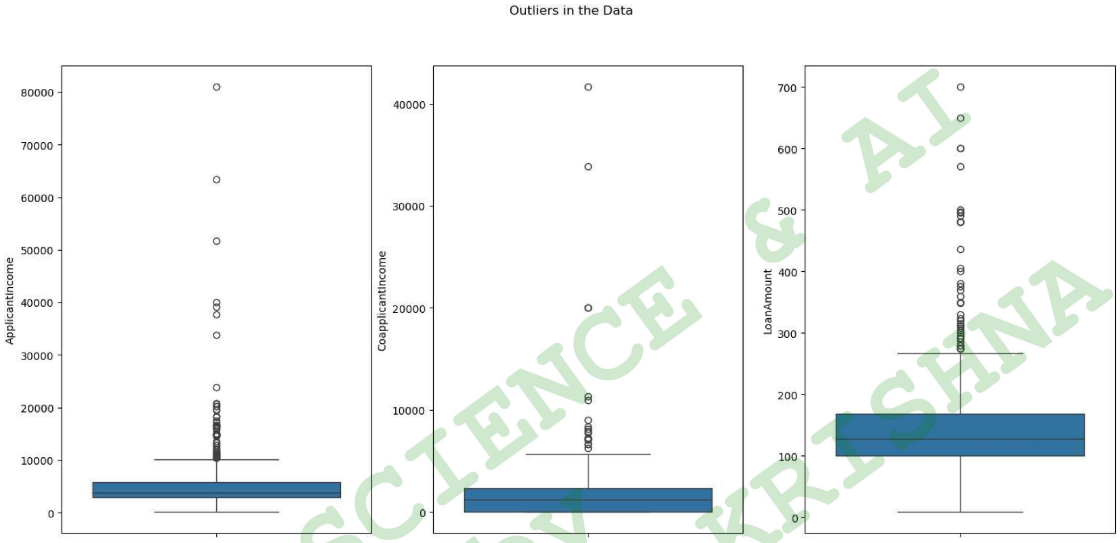


In [35]:
```
sns.heatmap(data[continous].corr(),annot=True)
plt.show()
```



In [36]:
```
# lets visualize the outliers using Box Plot

plt.subplot(1, 3, 1)
```

```
sns.boxplot(data['ApplicantIncome'])

plt.subplot(1, 3, 2)
sns.boxplot(data['CoapplicantIncome'])

plt.subplot(1, 3, 3)
sns.boxplot(data['LoanAmount'])

plt.suptitle('Outliers in the Data')
plt.show()
```



for Discrete Variables

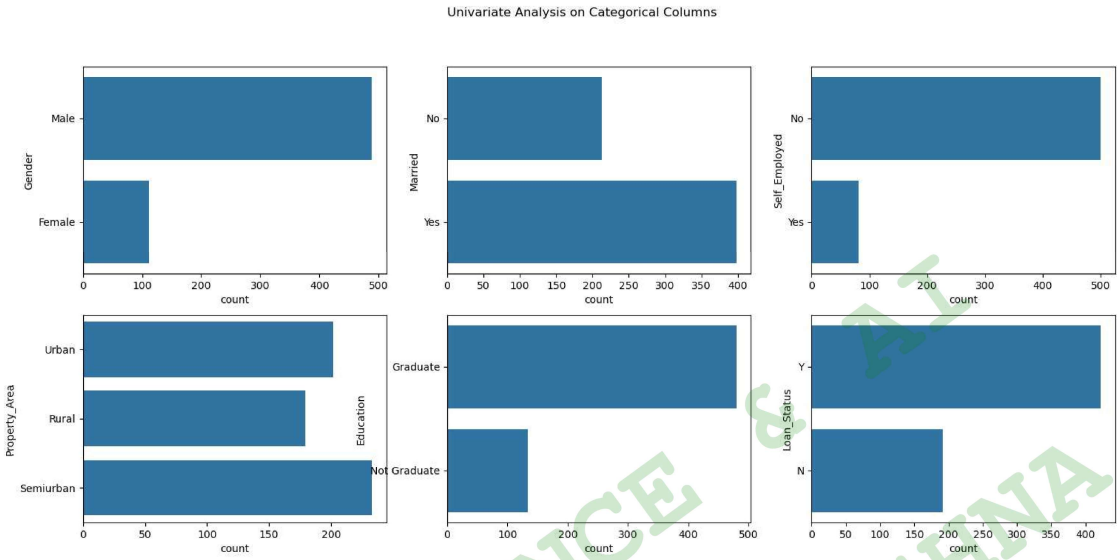
In [37]:
```
data[discrete_categorical].describe()
```

Out[37]:

	Gender	Married	Education	Self_Employed	Credit_History	Property_Area	Loan_Status
count	601	611	614	582	564	614	614
unique	2	2	2	2	2	3	2
top	Male	Yes	Graduate	No	good	Semiurban	Y
freq	489	398	480	500	475	233	422

In [38]:
```
plt.rcParams['figure.figsize'] = (18,8)

plt.subplot(2, 3, 1)
sns.countplot(data['Gender'])

plt.subplot(2, 3, 2)
sns.countplot(data['Married'])

plt.subplot(2, 3, 3)
sns.countplot(data['Self_Employed'])

plt.subplot(2, 3, 4)
sns.countplot(data['Property_Area'])

plt.subplot(2, 3, 5)
sns.countplot(data['Education'])

plt.subplot(2, 3, 6)
```

```
sns.countplot(data['Loan_Status'])

plt.suptitle('Univariate Analysis on Categorical Columns')
plt.show()
```



Data Preparation

In [39]:
```
data["Income"] = data['ApplicantIncome'] + data['CoapplicantIncome']

data.drop(columns=['ApplicantIncome','CoapplicantIncome'],inplace=True)
```

modifying the wrong data

In [40]:
```
data['Dependents']  = data['Dependents'].replace({'3+':3})
```

Missing Values Treatment

In [41]:
```
# checking no. of Missing values
data.isnull().sum()
```

Out[41]:
```
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
Property_Area        0
Loan_Status          0
Income               0
dtype: int64
```

In [42]:
```
# checking percentage of Missing values
data.isnull().sum()/ len(data)*100
```

Out[42]:
```
Gender              2.117264
Married             0.488599
Dependents          2.442997
Education           0.000000
Self_Employed       5.211726
LoanAmount          3.583062
Loan_Amount_Term    2.280130
Credit_History      8.143322
Property_Area       0.000000
Loan_Status         0.000000
Income              0.000000
dtype: float64
```

In [43]:
```
data = data.dropna(subset=["Income",'LoanAmount','Loan_Amount_Term','Credit_History
```

In [44]:
```
#count variable replace with 0
data['Dependents'] = data['Dependents'].fillna(0)
```

In [45]:
```
#categorical variables replace with mode
data['Gender'] = data['Gender'].fillna(data['Gender'].mode()[0])
data['Married'] = data['Married'].fillna(data['Married'].mode()[0])
data['Self_Employed'] = data['Self_Employed'].fillna(data['Self_Employed'].mode()[0
```

In [46]:
```
data.isnull().sum()
```

Out[46]:
```
Gender              0
Married             0
Dependents          0
Education           0
Self_Employed       0
LoanAmount          0
Loan_Amount_Term    0
Credit_History      0
Property_Area       0
Loan_Status         0
Income              0
dtype: int64
```

outliers treatment

Encoding

In [47]:
```
data['Gender'] = data['Gender'].map({'Male':1,'Female':0}).astype('int')
data['Married'] = data['Married'].map({'Yes':1,'No':0}).astype('int')
data['Education'] = data['Education'].map({'Graduate':1,'Not Graduate':0}).astype('
data['Self_Employed'] = data['Self_Employed'].map({'Yes':1,'No':0}).astype('int')
data['Property_Area'] = data['Property_Area'].map({'Rural':0,'Semiurban':1,'Urban':
data['Credit_History'] = data['Credit_History'].map({'good':1,'bad':0}).astype('int
data['Loan_Status'] = data['Loan_Status'].map({'Y':1,'N':0}).astype('int')
```

data type conversion

In [48]:
```
data['Dependents']        = data['Dependents'].astype('int')
data['Loan_Amount_Term']  = data['Loan_Amount_Term'].astype('int')
```

transformations

In [49]:
```
data[['Income','LoanAmount']].skew()
```

Out[49]:
```
Income        5.777628
LoanAmount    2.607945
dtype: float64
```

In [50]:
```
# lets apply boxcox transformation to remove skewness
from scipy.stats import boxcox
data['Income'],a = boxcox(data['Income'])
data['LoanAmount'],c = boxcox(data['LoanAmount'])
```

In [51]:
```
data[['Income','LoanAmount']].skew()
```

Out[51]:
```
Income       -0.027769
LoanAmount    0.038289
dtype: float64
```

In [52]:
```
data['Loan_Amount_Term']  = data['Loan_Amount_Term']/12
```

X&y

In [53]:
```
X = data.drop('Loan_Status',axis=1)
y = data['Loan_Status']
```

identify the best random state number

In [54]:
```
Train = []
Test = []
CV = []

for i in range(0, 101):
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

    from sklearn.linear_model import LogisticRegression
    log_default = LogisticRegression()
    log_default.fit(X_train,y_train)

    ypred_train = log_default.predict(X_train)
    ypred_test = log_default.predict(X_test)

    from sklearn.metrics import accuracy_score
    Train.append(accuracy_score(y_train, ypred_train))
    Test.append(accuracy_score(y_test, ypred_test))

    from sklearn.model_selection import cross_val_score
    CV.append(cross_val_score(log_default, X_train, y_train, cv=5, scoring="accurac

em = pd.DataFrame({"Train":Train, "Test":Test, "CV":CV})
gm = em[(abs(em['Train']-em['Test'])<=0.05) & (abs(em['Test']-em['CV'])<=0.05)]
rs = gm[gm["CV"]==gm["CV"].max()].index.to_list()[0]
print("best random_state number:",rs)
```

```
best random_state number: 70
```

train-test split

In [55]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.2, random_st
```

Machine Learning Modelling & Evaluation

1.Logistic Regression

In [56]:
```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression()
log_model.fit(X_train,y_train)

ypred_train = log_model.predict(X_train)
ypred_test  = log_model.predict(X_test)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Cross validation score :", cross_val_score(log_model,X_train,y_train,cv=5,sc
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.8226950354609929
Cross validation score : 0.8227731092436976
Test Accuracy : 0.7830188679245284
```

2. KNN

In [57]:
```
from sklearn.neighbors import KNeighborsClassifier

estimator = KNeighborsClassifier()
param_grid = {'n_neighbors': list(range(1,50))}

from sklearn.model_selection import GridSearchCV
knn_grid = GridSearchCV(estimator, param_grid,scoring='accuracy',cv=5)
knn_grid.fit(X_train,y_train)

knn_model = knn_grid.best_estimator_

ypred_train = knn_model.predict(X_train)
ypred_test  = knn_model.predict(X_test)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Cross validation score :", cross_val_score(knn_model,X_train,y_train,cv=5,sc
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.7541371158392435
Cross validation score : 0.7375910364145659
Test Accuracy : 0.7075471698113207
```

3.Support Vector Machine (SVM)

In [58]:
```
from sklearn.svm import SVC

estimator = SVC()
param_grid = {'C':[0.01,0.1,1],'kernel':['linear','rbf','sigmoid','poly']}

from sklearn.model_selection import GridSearchCV
svm_grid = GridSearchCV(estimator, param_grid,scoring='accuracy',cv=5)
svm_grid.fit(X_train,y_train)

svm_model = svm_grid.best_estimator_

ypred_train = svm_model.predict(X_train)
ypred_test  = svm_model.predict(X_test)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Cross validation score :", cross_val_score(svm_model,X_train,y_train,cv=5,sc
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.8226950354609929
Cross validation score : 0.8227731092436976
Test Accuracy : 0.7830188679245284
```

4.Decision Tree Classifier

In [59]:
```
from sklearn.tree import DecisionTreeClassifier
estimator = DecisionTreeClassifier(random_state=rs)
param_grid = {"criterion":["gini", "entropy"],
              "max_depth":list(range(1,16))}

from sklearn.model_selection import GridSearchCV
dt_grid = GridSearchCV(estimator,param_grid, scoring='accuracy',cv=5)
dt_grid.fit(X_train,y_train)

#identify the best model
dt = dt_grid.best_estimator_

#identify the importance of each feature
dt_fi = dt.feature_importances_

#identify the features where the feature importance is greater than 0
index = [i for i,x in enumerate(dt_fi) if x>0]

#crete new dataset with important features
X_train_dt = X_train.iloc[:,index]
X_test_dt = X_test.iloc[:,index]

#train with best model & with important features
dt.fit(X_train_dt,y_train)

ypred_train = dt.predict(X_train_dt)
ypred_test  = dt.predict(X_test_dt)

#Evaluate the best model
print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Cross validation score :", cross_val_score(dt,X_train_dt,y_train,cv=5,scorir
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.8226950354609929
Cross validation score : 0.8227731092436976
Test Accuracy : 0.7830188679245284
```

In [60]:
```
dt_grid.best_estimator_
```

Out[60]:
```
DecisionTreeClassifier(max_depth=1, random_state=70)
```

In [61]:
```
X_train_dt
```

Out[61]:

	Credit_History
158	1
119	1
448	0
519	1
391	1
...	...
70	1
370	1
140	1
252	1
390	1

423 rows × 1 columns

5.Random Forest Classifier

In [62]:
```
from sklearn.ensemble import RandomForestClassifier
estimator = RandomForestClassifier(random_state=rs)
param_grid = {'n_estimators':list(range(1,51))}

from sklearn.model_selection import GridSearchCV
rf_grid = GridSearchCV(estimator,param_grid, scoring="accuracy",cv=5)
rf_grid.fit(X_train,y_train)

rf = rf_grid.best_estimator_
rf_fi = rf.feature_importances_

index =[i for i,x in enumerate(rf_fi) if x>0]

X_train_rf = X_train.iloc[:,index]
X_test_rf = X_test.iloc[:,index]

rf.fit(X_train_rf,y_train)

ypred_train = rf.predict(X_train_rf)
ypred_test  = rf.predict(X_test_rf)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Cross validation score :", cross_val_score(rf,X_train_rf,y_train,cv=5,scorir
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.9905437352245863
Cross validation score : 0.8063025210084034
Test Accuracy : 0.7641509433962265
```

6. AdaBoost Classifier

In [63]:
```
from sklearn.ensemble import AdaBoostClassifier
estimator = AdaBoostClassifier(random_state=rs)
param_grid = {'n_estimators':list(range(1,51))}

from sklearn.model_selection import GridSearchCV
ab_grid = GridSearchCV(estimator,param_grid, scoring="accuracy",cv=5)
```

```
ab_grid.fit(X_train,y_train)

ab = ab_grid.best_estimator_
ab_fi = ab.feature_importances_

index =[i for i,x in enumerate(ab_fi) if x>0]

X_train_ab = X_train.iloc[:,index]
X_test_ab = X_test.iloc[:,index]

ab.fit(X_train_ab,y_train)

ypred_train = ab.predict(X_train_ab)
ypred_test  = ab.predict(X_test_ab)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Cross validation score :", cross_val_score(ab,X_train_ab,y_train,cv=5,scorin
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.83451536643026
Cross validation score : 0.8298879551820729
Test Accuracy : 0.7641509433962265
```

7. Gradient Boost Classifier

In [64]:

```
from sklearn.ensemble import GradientBoostingClassifier
estimator = GradientBoostingClassifier(random_state=rs)
param_grid = {"n_estimators": list(range(1,10)),"learning_rate":[0.1,0.2,0.3,0.4,0.

from sklearn.model_selection import GridSearchCV
gb_grid = GridSearchCV(estimator,param_grid, scoring="accuracy",cv=5)
gb_grid.fit(X_train,y_train)

gb = gb_grid.best_estimator_
gb_fi = gb.feature_importances_

index =[i for i,x in enumerate(gb_fi) if x>0]

X_train_gb = X_train.iloc[:,index]
X_test_gb = X_test.iloc[:,index]

gb.fit(X_train_gb,y_train)

ypred_train = gb.predict(X_train_gb)
ypred_test  = gb.predict(X_test_gb)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Cross validation score :", cross_val_score(gb,X_train_gb,y_train,cv=5,scorin
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.8392434988179669
Cross validation score : 0.8203921568627452
Test Accuracy : 0.7735849056603774
```

8. XGBoost Classifier

In [65]:

```
from xgboost import XGBClassifier
estimator = XGBClassifier(random_state=rs)
param_grid = {"n_estimators":[10,20,40,100],'max_depth':[3,4,5],'gamma':[0,0.15,0.3

from sklearn.model_selection import GridSearchCV
xgb_grid = GridSearchCV(estimator,param_grid, scoring="accuracy",cv=5)
xgb_grid.fit(X_train,y_train)
```

```
xgb = xgb_grid.best_estimator_

xgb_fi = xgb.feature_importances_

index =[i for i,x in enumerate(xgb_fi) if x>0]

X_train_xgb = X_train.iloc[:,index]
X_test_xgb = X_test.iloc[:,index]

xgb.fit(X_train_xgb,y_train)

ypred_train = xgb.predict(X_train_xgb)
ypred_test  = xgb.predict(X_test_xgb)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("Cross validation score :", cross_val_score(xgb,X_train_xgb,y_train,cv=5,scor
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.8557919621749409
Cross validation score : 0.8275070028011206
Test Accuracy : 0.7735849056603774
```