```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
drive.mount('/content/drive')
df=pd.read_csv('/content/drive/MyDrive/loan_data.csv')
df.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 45000,\n  \"fields\":
[\n    {\n      \"column\": \"person_age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 6.045108211348622,\n
\"min\": 20.0,\n        \"max\": 144.0,\n
\"num_unique_values\": 60,\n        \"samples\": [\n          22.0,\n
26.0,\n          53.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"person_gender\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"male\",\n          \"female\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"person_education\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"High
School\",\n          \"Doctorate\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"person_income\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
80422.49863189556,\n        \"min\": 8000.0,\n        \"max\":
7200766.0,\n        \"num_unique_values\": 33989,\n
\"samples\": [\n          48967.0,\n          31001.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"person_emp_exp\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6,\n        \"min\": 0,\n        \"max\": 125,\n
\"num_unique_values\": 63,\n        \"samples\": [\n          93,\n
76\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"person_home_ownership\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          \"OWN\",\n          \"OTHER\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"loan_amnt\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6314.8866905411405,\n        \"min\": 500.0,\n        \"max\":
35000.0,\n        \"num_unique_values\": 4483,\n        \"samples\":
[\n          5800.0,\n          28338.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"loan_intent\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 6,\n        \"samples\": [\n
\"PERSONAL\",\n          \"EDUCATION\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"loan_int_rate\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.9788082802254734,\n        \"min\": 5.42,\n        \"max\": 20.0,\n
\"num_unique_values\": 1302,\n        \"samples\": [\n          15.0,\
n          13.45\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"loan_percent_income\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.08721230801403355,\n        \"min\":
0.0,\n        \"max\": 0.66,\n        \"num_unique_values\": 64,\n
\"samples\": [\n          0.45,\n          0.54\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cb_person_cred_hist_length\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3.8797018451620433,\n        \"min\": 2.0,\n        \"max\": 30.0,\n
\"num_unique_values\": 29,\n        \"samples\": [\n          24.0,\n
25.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"credit_score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 50,\n        \"min\": 390,\n
\"max\": 850,\n        \"num_unique_values\": 340,\n
\"samples\": [\n          492,\n          484\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"previous_loan_defaults_on_file\",\
n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"Yes\",\
n          \"No\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"loan_status\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


DATA PREPROCESSING AND CLEANING

```
df.shape
```

```
(45000, 14)
```

```
df.duplicated().sum()
```

```
np.int64(0)
```

```
df.isnull().sum()
```

```
person_age                          0
person_gender                       0
person_education                    0
person_income                       0
person_emp_exp                      0
person_home_ownership               0
loan_amnt                           0
loan_intent                         0
loan_int_rate                       0
loan_percent_income                 0
cb_person_cred_hist_length          0
credit_score                        0
previous_loan_defaults_on_file      0
loan_status                         0
dtype: int64
```

```
df.describe()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n{\n      \"column\": \"person_age\",\n      \"properties\": {\n\"dtype\": \"number\",\n        \"std\": 15895.92981900713,\n\"min\": 6.045108211348622,\n        \"max\": 45000.0,\n\"num_unique_values\": 8,\n        \"samples\": [\n27.76417777777778,\n          26.0,\n          45000.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"person_income\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2524599.5465676305,\n        \"min\": 8000.0,\n        \"max\": 7200766.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          80319.05322222222,\n          67048.0,\n          45000.0\n],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n}\n    },\n    {\n      \"column\": \"person_emp_exp\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 15902.409513060764,\n        \"min\": 0.0,\n        \"max\": 45000.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n5.410333333333333,\n          4.0,\n          45000.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"loan_amnt\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 15907.41915594131,\n        \"min\": 500.0,\n        \"max\": 45000.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n9583.157555555556,\n          8000.0,\n          45000.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"loan_int_rate\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 15906.267079441906,\n        \"min\": 2.9788082802254734,\n\"max\": 45000.0,\n        \"num_unique_values\": 8,\n\"samples\": [\n          11.006605777777779,\n          11.01,\n45000.0\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"loan_percent_income\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 15909.83858800236,\n        \"min\":
0.0,\n        \"max\": 45000.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          0.1397248888888889,\n          0.12,\n
45000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cb_person_cred_hist_length\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 15907.039026136903,\n
\"min\": 2.0,\n        \"max\": 45000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
5.8674888888888885,\n          4.0,\n          45000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"credit_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
15718.058537708299,\n        \"min\": 50.435865000741984,\n
\"max\": 45000.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          632.6087555555556,\n          640.0,\n
45000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"loan_status\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 15909.819850659878,\n        \"min\":
0.0,\n        \"max\": 45000.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.2222222222222222,\n          1.0,\n
0.41574432904844355\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
df.columns
```

```
Index(['person_age', 'person_gender', 'person_education',
'person_income',
       'person_emp_exp', 'person_home_ownership', 'loan_amnt',
'loan_intent',
       'loan_int_rate', 'loan_percent_income',
'cb_person_cred_hist_length',
       'credit_score', 'previous_loan_defaults_on_file',
'loan_status'],
      dtype='object')
```

REMOVING UNNECESSARY COLUMNS AND CHECKING UNIQUE COLUMNS

```
df.drop(columns=['person_age', 'person_gender','person_education',
'person_emp_exp','loan_percent_income'],inplace=True)

df.rename(columns={'person_income':'Income','person_home_ownership':'O
wnership','loan_amnt':'Loan Amount','loan_intent':'Loan Intent
','loan_int_rate':'Interest
Rate','cb_person_cred_hist_length':'Credit.Hist','credit_score':'Cr.Sc
ore','previous_loan_defaults_on_file':'Previous
Loans','loan_status':'Approval'},inplace=True)
```

```
df.head(4)
df.describe()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n{\n      \"column\": \"Income\",\n      \"properties\": {\n\"dtype\": \"number\",\n        \"std\": 2524599.5465676305,\n\"min\": 8000.0,\n        \"max\": 7200766.0,\n\"num_unique_values\": 8,\n        \"samples\": [\n80319.05322222222,\n          67048.0,\n          45000.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n}\n    },\n    {\n      \"column\": \"Loan Amount\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 15907.41915594131,\n        \"min\": 500.0,\n        \"max\": 45000.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n9583.157555555556,\n          8000.0,\n          45000.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Interest Rate\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 15906.267079441906,\n        \"min\": 2.9788082802254734,\n\"max\": 45000.0,\n        \"num_unique_values\": 8,\n\"samples\": [\n          11.006605777777779,\n          11.01,\n45000.0\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Credit.Hist\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 15907.039026136903,\n        \"min\": 2.0,\n        \"max\": 45000.0,\n        \"num_unique_values\": 8,\n\"samples\": [\n          5.8674888888888885,\n          4.0,\n45000.0\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Cr.Score\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 15718.058537708299,\n        \"min\": 50.435865000741984,\n        \"max\": 45000.0,\n\"num_unique_values\": 8,\n        \"samples\": [\n632.6087555555556,\n          640.0,\n          45000.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Approval\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 15909.819850659878,\n        \"min\": 0.0,\n        \"max\": 45000.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\n0.2222222222222222,\n          1.0,\n          0.41574432904844355\n],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n}\n    }\n  ]\n}","type":"dataframe"}
```


```
df['Ownership'].unique()
```

```
array(['RENT', 'OWN', 'MORTGAGE', 'OTHER'], dtype=object)
```

```
df['Approval'].unique()
```

```
array([1, 0])
```

```
df['Previous Loans'].unique()
```

```
array(['No', 'Yes'], dtype=object)
```

```
df['Cr.Score'].unique()
```

```
array([561, 504, 635, 675, 586, 532, 701, 585, 544, 640, 621, 651,
573,
       708, 583, 670, 663, 694, 709, 679, 684, 662, 691, 600, 654,
626,
       607, 700, 553, 589, 681, 567, 669, 606, 582, 649, 602, 616,
631,
       637, 695, 620, 622, 645, 624, 570, 648, 652, 559, 623, 609,
579,
       688, 661, 562, 664, 564, 598, 557, 677, 690, 599, 604, 601,
634,
       671, 789, 538, 587, 683, 518, 617, 668, 673, 706, 536, 689,
595,
       584, 642, 614, 597, 625, 603, 643, 508, 505, 593, 686, 646,
697,
       615, 687, 650, 588, 658, 531, 665, 703, 594, 618, 574, 577,
653,
       630, 660, 639, 612, 628, 592, 580, 678, 672, 613, 566, 718,
484,
       699, 656, 659, 636, 554, 807, 578, 674, 608, 569, 629, 560,
548,
       667, 676, 581, 655, 551, 529, 666, 576, 633, 611, 657, 647,
542,
       692, 545, 540, 525, 537, 641, 539, 563, 712, 491, 590, 572,
528,
       638, 627, 596, 547, 507, 565, 693, 522, 632, 556, 499, 704,
503,
       714, 552, 555, 558, 521, 605, 571, 591, 719, 610, 535, 644,
523,
       546, 702, 711, 534, 805, 682, 447, 725, 680, 568, 524, 710,
707,
       619, 460, 696, 527, 511, 477, 575, 496, 685, 476, 502, 541,
722,
       506, 487, 530, 515, 513, 520, 724, 514, 501, 549, 720, 486,
716,
       465, 509, 517, 444, 482, 698, 443, 512, 526, 454, 550, 462,
717,
       727, 715, 543, 448, 713, 721, 481, 723, 533, 705, 483, 740,
495,
       459, 489, 519, 494, 498, 488, 468, 490, 479, 510, 472, 480,
421,
       473, 500, 493, 441, 485, 450, 467, 461, 475, 463, 728, 735,
458,
       456, 726, 729, 750, 455, 478, 453, 445, 431, 737, 497, 492,
437,
```

```
       516, 451, 434, 435, 439, 466, 440, 449, 469, 471, 730, 470,
734,
       751, 739, 736, 738, 733, 732, 731, 741, 748, 744, 762, 742,
745,
       743, 464, 390, 755, 747, 759, 746, 756, 753, 764, 850, 760,
754,
       770, 784, 773, 765, 772, 419, 474, 430, 418, 420, 792, 768,
457,
       446, 767])
```

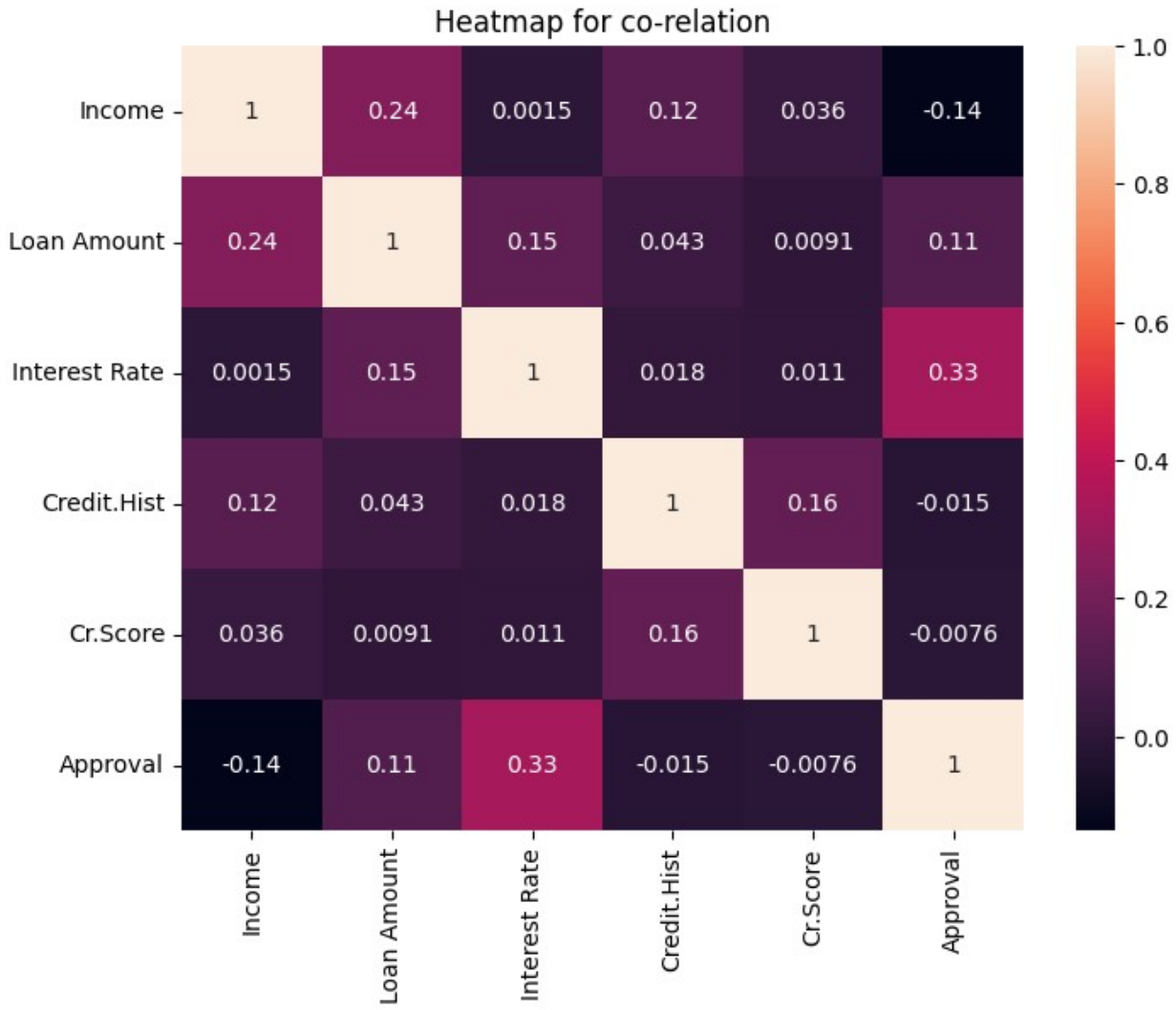
```
df['Credit.Hist'].unique()
```

```
array([ 3.,  2.,  4.,  8.,  7.,  6.,  9., 10.,  5., 11., 16., 15.,
12.,
       13., 17., 14., 25., 28., 27., 22., 19., 29., 23., 26., 20.,
21.,
       30., 24., 18.])
```

CHECKING CO-RELATION% BETWEEN COLUMNS

```
plt.figure(figsize=(8,6))
plt.title('Heatmap for co-relation')
sns.heatmap(df.corr(numeric_only=True),annot=True)
```

```
<Axes: title={'center': 'Heatmap for co-relation'}>
```



Loan amount has hogher degree of relation with income,and approval of loan has high degree relation with interest rate

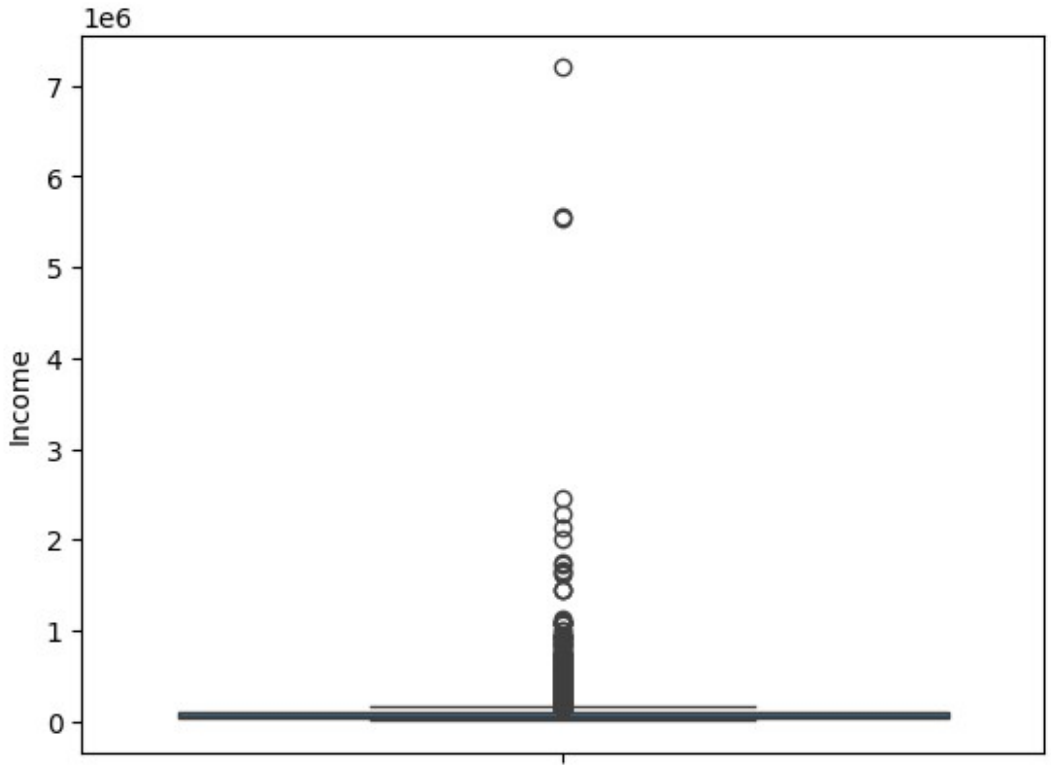
TRAIN_TEST_SPLIT_

```
df.columns
```

```
Index(['Income', 'Ownership', 'Loan Amount', 'Loan Intent ', 'Interest
Rate',
       'Credit.Hist', 'Cr.Score', 'Previous Loans', 'Approval'],
      dtype='object')
```

```
sns.boxplot(df['Income'])
```

```
<Axes: ylabel='Income'>
```




```
df['Income'].value_counts()
```

```
Income
8000.0     15
73011.0    10
36995.0     9
37020.0     8
60914.0     8
           ..
16720.0     1
29547.0     1
64117.0     1
89930.0     1
42960.0     1
Name: count, Length: 33989, dtype: int64
```

```
len(df[df['Income']>400000])
```

```
165
```

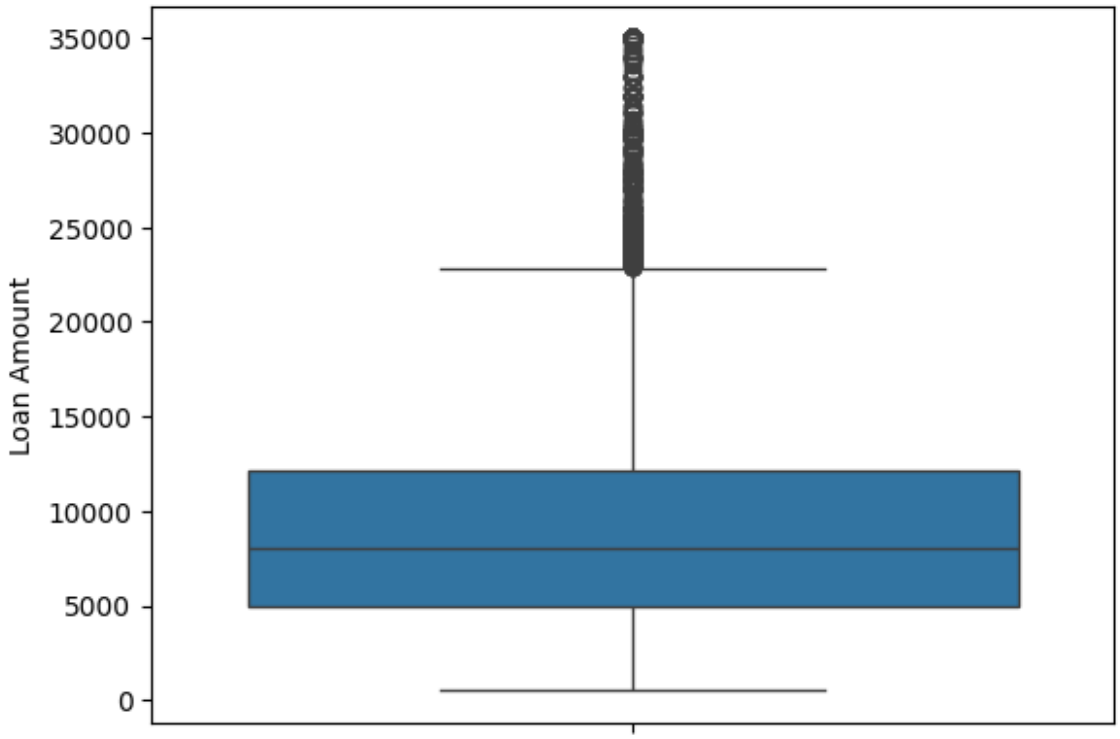
```
df=df[df['Income']<400000]
```

```
df.shape
```

```
(44835, 9)
```

```
sns.boxplot(df['Loan Amount'])
```

```
<Axes: ylabel='Loan Amount'>
```

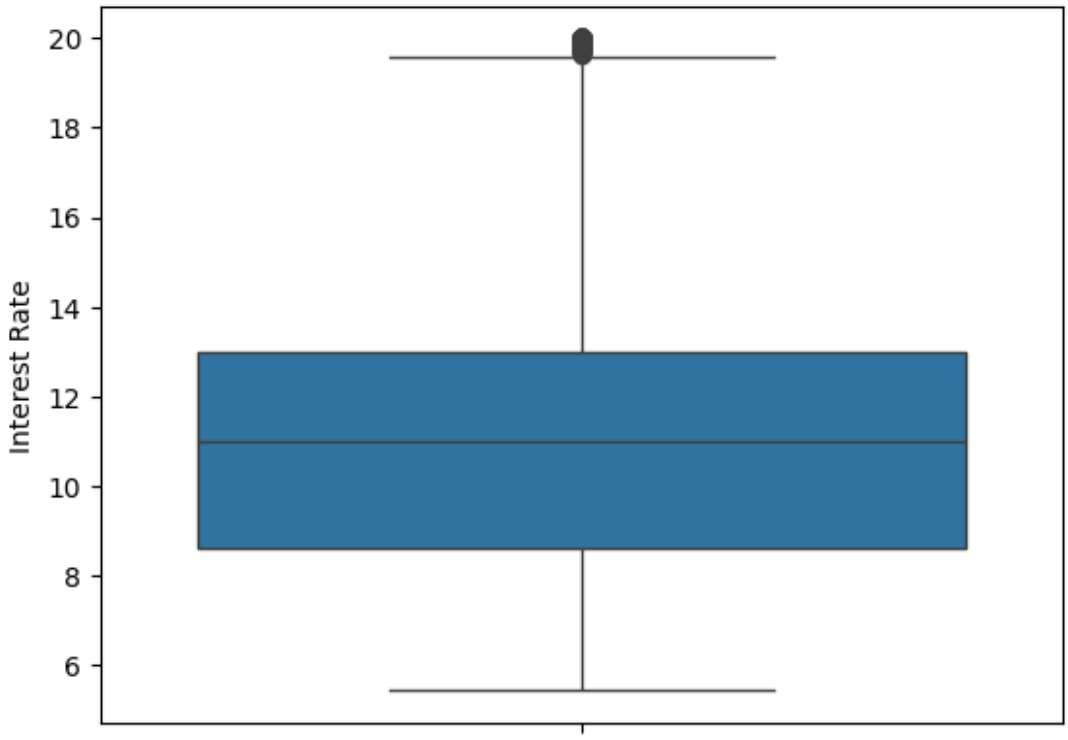



```
df['Loan Amount'].value_counts()
```

```
Loan Amount
10000.0    3607
5000.0     2786
6000.0     2423
12000.0    2412
15000.0    1999
           ...
6074.0        1
3108.0        1
2457.0        1
4601.0        1
17783.0       1
Name: count, Length: 4451, dtype: int64
```

```
sns.boxplot(df['Interest Rate'])
```

```
<Axes: ylabel='Interest Rate'>
```




```
df['Interest Rate'].value_counts()
```

```
Interest Rate
11.01    3318
10.99     802
7.51      796
7.49      686
7.88      671
         ...
19.25       1
19.62       1
19.11       1
19.80       1
16.92       1
Name: count, Length: 1302, dtype: int64
```

```
x=df.drop(columns=['Approval'])
x
```

```
{"summary":"{\n  \"name\": \"x\",\n  \"rows\": 44835,\n  \"fields\":
[\n    {\n      \"column\": \"Income\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 45588.23413018292,\n
\"min\": 8000.0,\n        \"max\": 397048.0,\n
\"num_unique_values\": 33824,\n        \"samples\": [\n
27049.0,\n          24652.0,\n          79904.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Ownership\",\n
```

\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n   \"samples\": [\n   \"OWN\",\
n   \"OTHER\",\n   \"RENT\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Loan Amount\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
6290.258971072717,\n   \"min\": 500.0,\n   \"max\":
35000.0,\n   \"num_unique_values\": 4451,\n   \"samples\":
[\n   17507.0,\n   14807.0,\n   6016.0\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Loan Intent \",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 6,\n   \"samples\":
[\n   \"PERSONAL\",\n   \"EDUCATION\",\n
\"DEBTCONSOLIDATION\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Interest Rate\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 2.9794284164692604,\n   \"min\":
5.42,\n   \"max\": 20.0,\n   \"num_unique_values\": 1302,\n
\"samples\": [\n   7.8,\n   8.12,\n   16.96\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Credit.Hist\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
3.859562852100272,\n   \"min\": 2.0,\n   \"max\": 30.0,\n
\"num_unique_values\": 29,\n   \"samples\": [\n   24.0,\n
27.0,\n   12.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Cr.Score\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 50,\n   \"min\": 390,\n
\"max\": 807,\n   \"num_unique_values\": 337,\n
\"samples\": [\n   652,\n   522,\n   557\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Previous Loans\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n   \"Yes\",\
n   \"No\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"x"}

```
y=df['Approval']
y
```

```
0        1
1        0
2        1
3        1
4        1
       ..
44995    1
44996    1
```

```
44997    1
44998    1
44999    1
Name: Approval, Length: 44835, dtype: int64
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,rando
m_state=42)
```

ENCODING DATA

```
from sklearn.preprocessing import OneHotEncoder
ohe=OneHotEncoder()
ohe.fit(x[['Ownership','Loan Intent ']])
```

```
OneHotEncoder()
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
x['Previous Loans']=le.fit_transform(x['Previous Loans'])
```

DATA SCALING

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x[['Interest Rate']])
scaler.fit(x[['Income']])
scaler.fit(x[['Loan Amount']])
```

```
StandardScaler()
```

```
from math import remainder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
column_transform=make_column_transformer((OneHotEncoder(categories=ohe
.categories_),['Income','Loan Intent']),
                                         (StandardScaler(),['Interest
Rate','Loan Amount','Income']),
                                          remainder='passthrough')
column_transform
```

```
ColumnTransformer(remainder='passthrough',
                  transformers=[('onehotencoder',

OneHotEncoder(categories=[array(['MORTGAGE', 'OTHER', 'OWN', 'RENT'],
dtype=object),

array(['DEBTCONSOLIDATION', 'EDUCATION', 'HOMEIMPROVEMENT', 'MEDICAL',
       'PERSONAL', 'VENTURE'], dtype=object)]),
                                  ['Income', 'Loan Intent']),
```

```
                                    ('standardscaler', StandardScaler(),
                                     ['Interest Rate', 'Loan Amount',
'Income'])])

from sklearn.preprocessing import OneHotEncoder, StandardScaler,
OrdinalEncoder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

# Define categorical columns for OneHotEncoder, numerical columns for
StandardScaler,
# and the binary categorical column for OrdinalEncoder.
categorical_cols_ohe = ['Ownership', 'Loan Intent ']
numerical_cols = ['Income', 'Loan Amount', 'Interest Rate',
'Credit.Hist', 'Cr.Score']
binary_categorical_col = ['Previous Loans'] # This column needs
explicit encoding

# Create the ColumnTransformer
column_transform = make_column_transformer(
    (OneHotEncoder(categories=ohe.categories_), categorical_cols_ohe),
    (StandardScaler(), numerical_cols),
    (OrdinalEncoder(), binary_categorical_col), # OrdinalEncoder will
convert 'Yes'/'No' to numerical
    remainder='drop' # Explicitly drop any unhandled columns
)

x_train_trans=column_transform.fit_transform(x_train)
x_test_trans=column_transform.transform(x_test)
```

MODELING

```
#LINEAR REGRESSION (changed to Logistic Regression for classification)
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(max_iter=1000) # Increased max_iter for
convergence

#TREE BASED REGRESSION (changed to Decision Tree Classifier)
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()

#DISTANCE BASED MODELS (changed to K-Nearest Neighbors Classifier)
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()

#RANDOM FOREST CLASSIFIER
from sklearn.ensemble import RandomForestClassifier # Changed from
Regressor to Classifier
rfc=RandomForestClassifier()
```

```
#BAGGING CLASSIFIER
from sklearn.ensemble import BaggingClassifier
bgc=BaggingClassifier()

#EXTRA TREE CLASSIFIER
from sklearn.ensemble import ExtraTreesClassifier
etc=ExtraTreesClassifier()

#XGB CLASSIFIER
from xgboost import XGBClassifier
xgb=XGBClassifier()

#METRICS
from sklearn.metrics import
accuracy_score,precision_score,recall_score,f1_score,confusion_matrix,
ConfusionMatrixDisplay
```

Linear Regression: It imports LinearRegression for linear modeling. Tree-Based Regression: It imports DecisionTreeRegressor for decision tree-based regression tasks. Distance-Based Models: KNeighborsRegressor is imported, which is a distance-based algorithm for regression. Ensemble Models: It imports several ensemble methods: RandomForestRegressor, BaggingClassifier, and ExtraTreesClassifier. These models combine predictions from multiple base estimators to improve robustness and accuracy. XGBoost Classifier: XGBClassifier is imported from the xgboost library, which is a powerful gradient boosting framework for classification. Metrics: Finally, it imports a comprehensive set of evaluation metrics from sklearn.metrics, including accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, and ConfusionMatrixDisplay. These will be used to assess the performance of the trained models.

```
classifiers={
    'LR':lr,
    'DT':dt,
    'KNN':knn,
    'RFC':rfc,
    'XGB':xgb
}


def train_classifier(classifier,x_train,x_test,y_train,y_test):

  #TRAINING CLASSIFIER
  classifier.fit(x_train,y_train)

  #MAKING PREDICTIONS
  y_pred=classifier.predict(x_test)

  #CALCULATING METRICS
  accuracy=accuracy_score(y_test,y_pred)
  precision=precision_score(y_test,y_pred,average='weighted')
  recall=recall_score(y_test,y_pred,average='weighted')
```

```
  f1=f1_score(y_test , y_pred, average='weighted')

  #GENERATING AND DISPLAYING CONFUSION MATRIX
  cm=confusion_matrix(y_test,y_pred)
  disp=ConfusionMatrixDisplay(confusion_matrix=cm)
  disp.plot(cmap=plt.cm.Blues)


  plt.title(f'Confusion Matrix for {name}')
  plt.xlabel("PREDICTED LABELS")
  plt.ylabel("TRUE LABELS")
  plt.show()

  return accuracy,precision,recall,f1
```

The code in this cell defines a Python function called train_classifier which automates the process of training a classification model, making predictions, evaluating its performance, and visualizing the results. Let's break it down:

Function Definition: The function train_classifier takes five arguments: classifier (the machine learning model you want to train), x_train and y_train (your training data and labels), and x_test and y_test (your testing data and true labels). Training Classifier: classifier.fit(x_train, y_train) trains the provided classifier model using the training features (x_train) and their corresponding true labels (y_train). The model learns patterns from this data. Making Predictions: y_pred=classifier.predict(x_test) uses the now-trained model to make predictions on the unseen test features (x_test). The results are stored in y_pred. Calculating Metrics: It then calculates several key classification metrics: accuracy_score: The proportion of correctly classified instances. precision_score: The ability of the classifier not to label as positive a sample that is negative (weighted average for multi-class). recall_score: The ability of the classifier to find all the positive samples (weighted average for multi-class). f1_score: The harmonic mean of precision and recall (weighted average for multi-class). Generating and Displaying Confusion Matrix: A confusion_matrix is created to summarize the performance of a classification algorithm. ConfusionMatrixDisplay then visualizes this matrix, showing the counts of true positive, true negative, false positive, and false negative predictions. Plot Customization: It sets a title for the confusion matrix plot, labels the x and y axes as 'PREDICTED LABELS' and 'TRUE LABELS' respectively, and finally plt.show() displays the plot. Return Values: The function returns the calculated accuracy, precision, recall, and f1 scores.

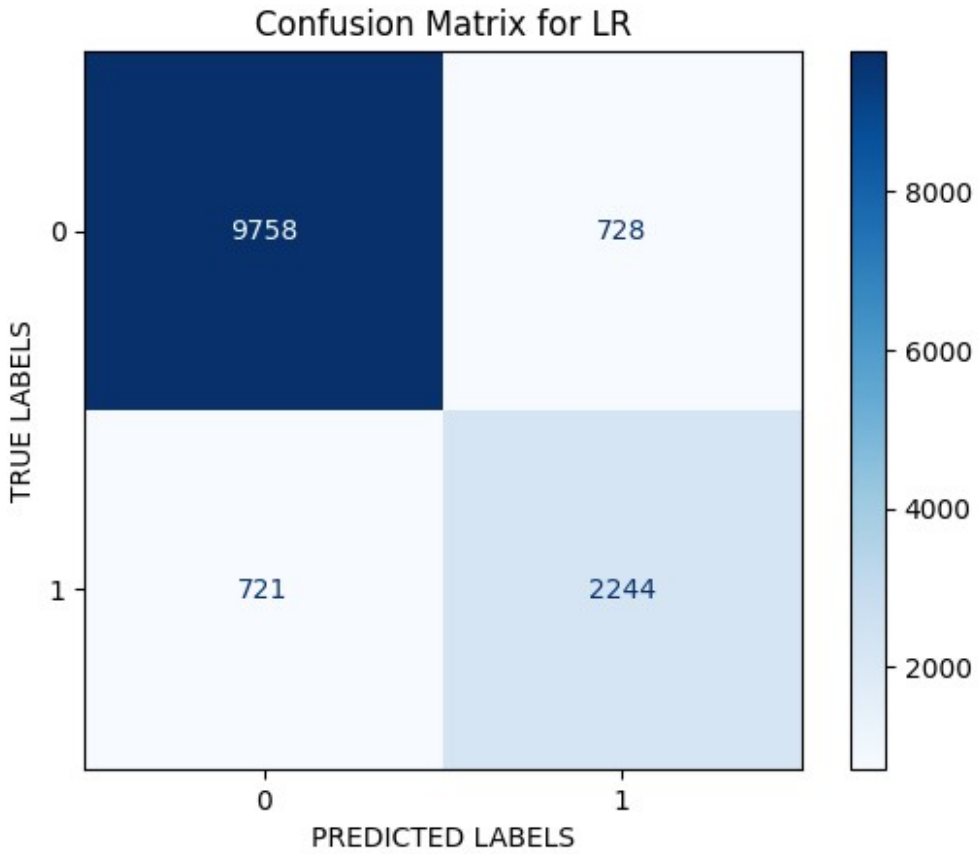
```
# STORING ACCURACY,PRECISION,RECALL,AND F1 SCORES
accuracy_scores=[]
precision_scores=[]
recall_scores=[]
f1_scores=[]

#LOOP THROUGH EACH CLASSIFIER
for name,clf in classifiers.items():
    print(f'\nTraining {name}...')
    current_accuracy, current_precision, current_recall, current_f1 =
train_classifier(clf, x_train_trans, x_test_trans, y_train, y_test)
```

```
    # PRINT THE CLASSIFIER'S PERFORMANCE MATRIX
    print(f'Name: {name}')
    print(f'Accuracy: {current_accuracy}')
    print(f'Precision: {current_precision}')
    print(f'Recall: {current_recall}')
    print(f'F1 Score: {current_f1}')
    print('-----------------------------')

    # APPEND SCORES TO LIST
    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
    recall_scores.append(current_recall)
    f1_scores.append(current_f1)
```

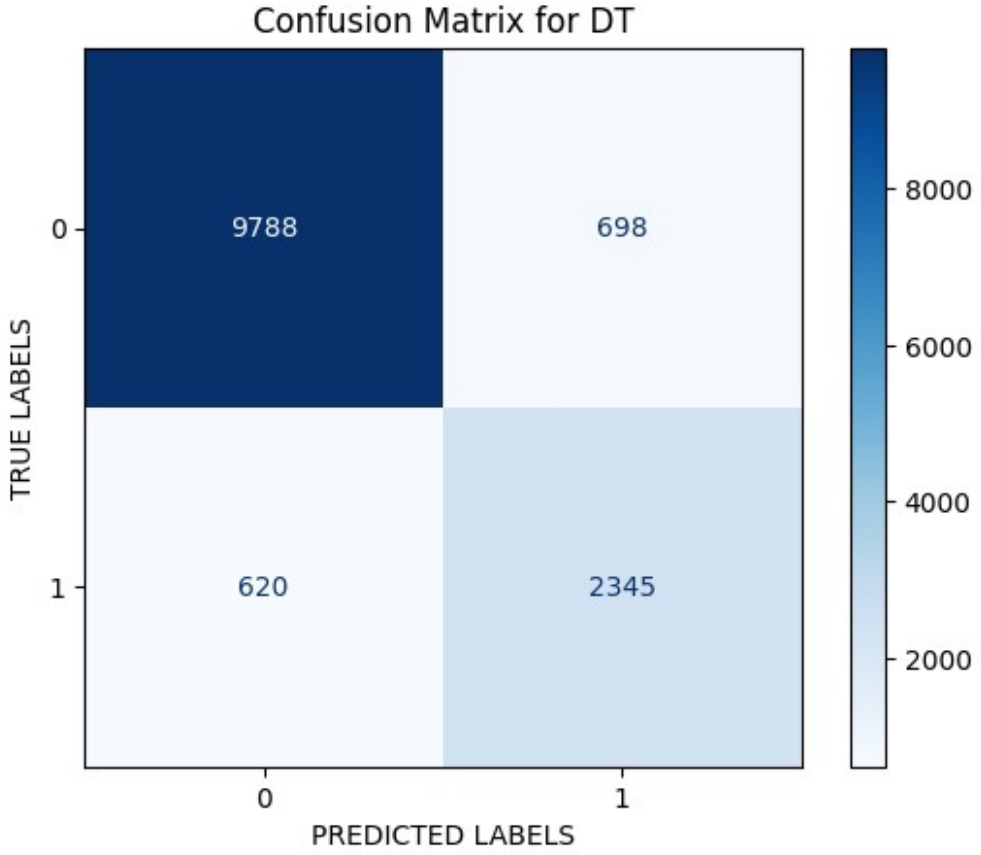
```
Training LR...
```




```
Name: LR
Accuracy: 0.8922756672366366
Precision: 0.8923673362820014
Recall: 0.8922756672366366
F1 Score: 0.8923211892149823
```

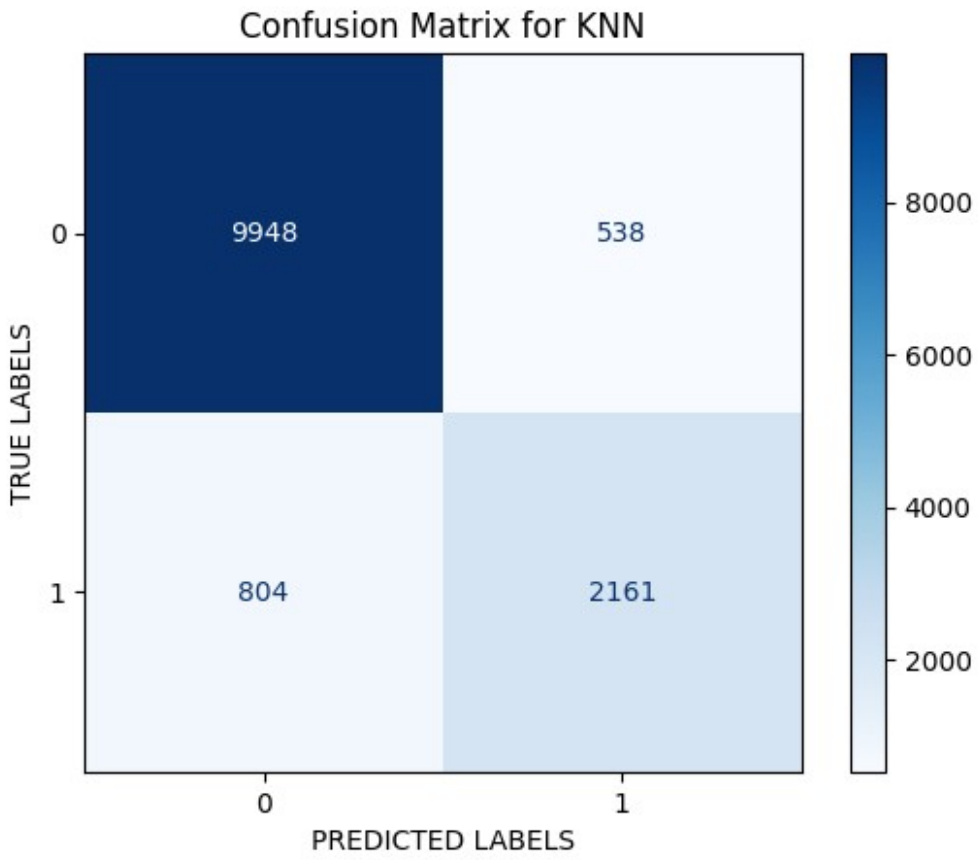
```
----------------------------

Training DT...
```



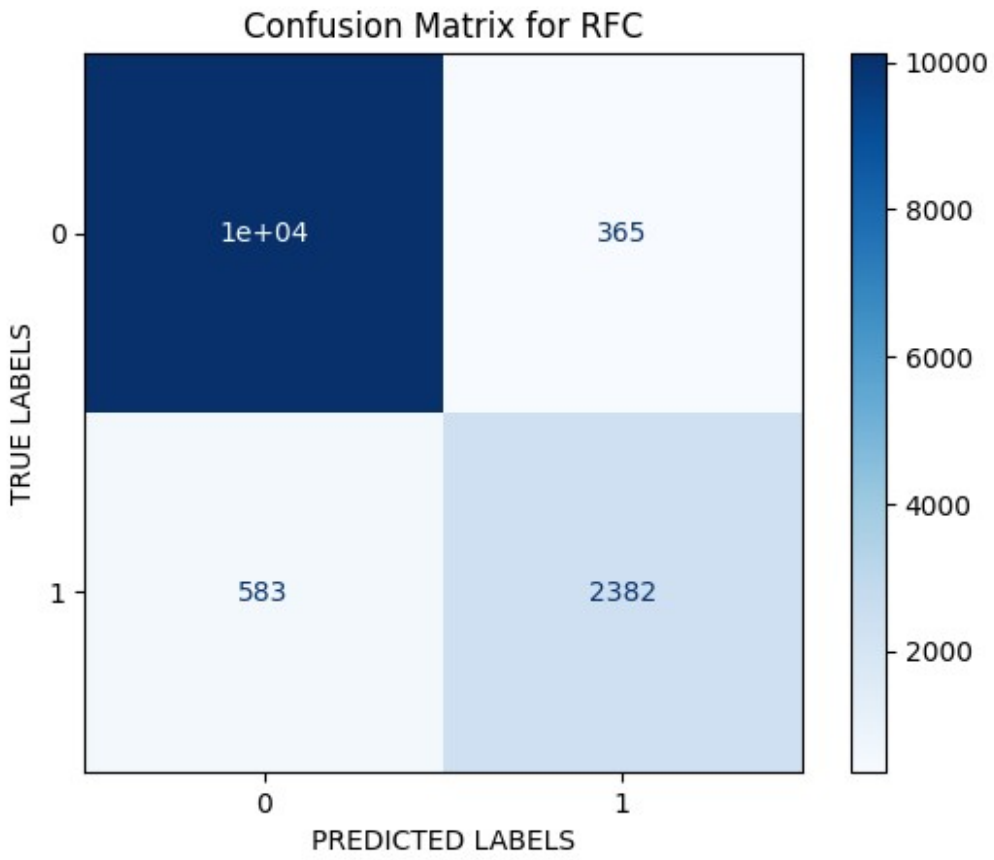

```
Name: DT
Accuracy: 0.9020147200951603
Precision: 0.9029994148063958
Recall: 0.9020147200951603
F1 Score: 0.9024678804882862
----------------------------

Training KNN...
```



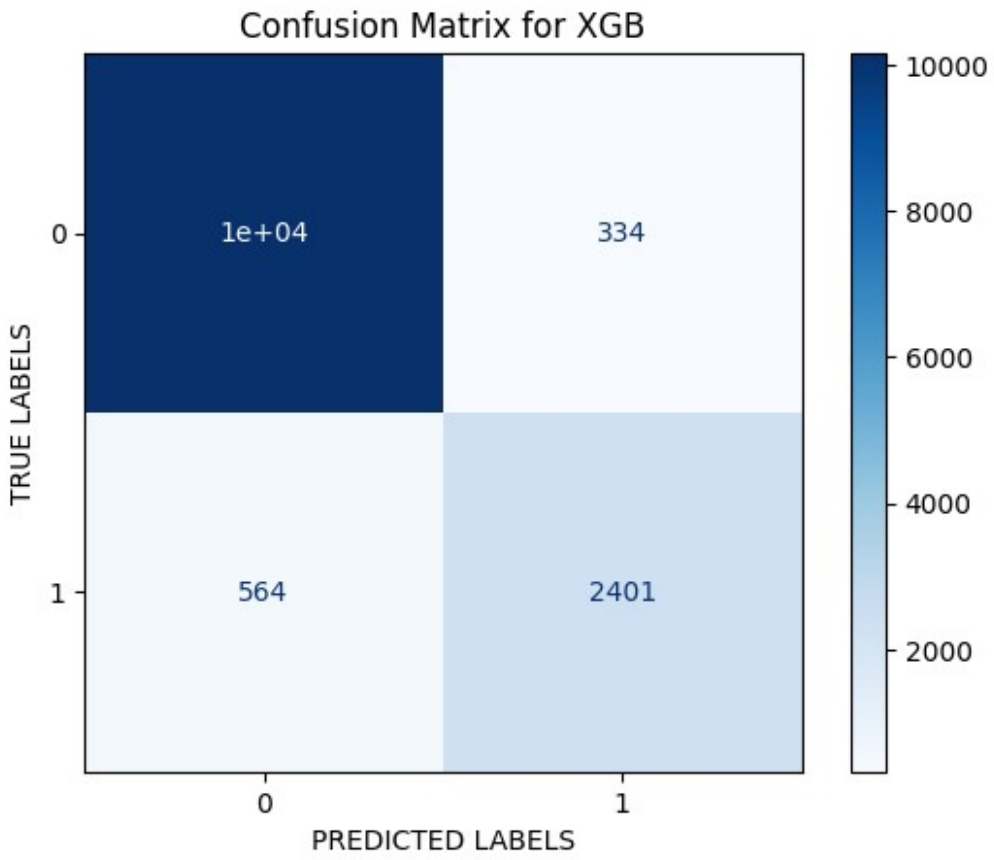

```
Name: KNN
Accuracy: 0.9002304661363467
Precision: 0.8977673053331857
Recall: 0.9002304661363467
F1 Score: 0.8985125067472145
----------------------------

Training RFC...
```




```
Name: RFC
Accuracy: 0.9295219686268679
Precision: 0.9282512348781771
Recall: 0.9295219686268679
F1 Score: 0.9285395966858785
----------------------------

Training XGB...
```




```
Name: XGB
Accuracy: 0.9332391643743959
Precision: 0.932050963629974
Recall: 0.9332391643743959
F1 Score: 0.9322543471153916
----------------------------
```

First, it sets up four empty lists: accuracy_scores, precision_scores, recall_scores, and f1_scores. These lists will store the performance metrics for each classifier after it's been trained and evaluated.

Looping Through Classifiers: The for name, clf in classifiers.items(): loop iterates through each classifier in your classifiers dictionary. In each iteration:

name will be the string key (e.g., 'LR', 'DT', 'XGB'). clf will be the actual classifier object (e.g., LogisticRegression(), DecisionTreeClassifier()). Training and Evaluation: Inside the loop, it calls the train_classifier function (which was defined in the previous cell). This function:

Trains the current clf using x_train_trans and y_train. Makes predictions on x_test_trans. Calculates the accuracy, precision, recall, and f1_score using y_test and the predictions. Displays a confusion matrix for the current classifier. Returns these four performance metrics. Printing Results: After the train_classifier function returns the scores, the code prints the Name of the

classifier and its calculated Accuracy, Precision, Recall, and F1 Score to the console, providing a clear overview of how each model performed.

Storing Scores: Finally, accuracy_scores.append(current_accuracy) (and similarly for precision, recall, and f1) adds the scores of the current classifier to their respective lists. This accumulates the results, allowing you to later compare the performance of all classifiers collectively.

```
results = {name: acc for name, acc in zip(classifiers.keys(),
accuracy_scores)}
best_model_name = max(results, key=results.get)
print("Best Model:", best_model_name)
```

```
Best Model: XGB
```

TRAINING PIPELINE

```
from sklearn.pipeline import Pipeline

final_model = Pipeline(steps=[
    ("preprocessing", column_transform),
    ("model", classifiers[best_model_name])
])
final_model.fit(x_train,y_train)
```

```
Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('onehotencoder',

OneHotEncoder(categories=[array(['MORTGAGE', 'OTHER', 'OWN', 'RENT'],
dtype=object),

array(['DEBTCONSOLIDATION', 'EDUCATION', 'HOMEIMPROVEMENT', 'MEDICAL',
       'PERSONAL', 'VENTURE'], dtype=object)]),
                                                  ['Ownership',
                                                   'Loan Intent ']),
                                                 ('standardscaler',
                                                  StandardScaler(),
                                                  ['Income', 'Loan
Amoun...
                                feature_types=None,
feature_weights=None,
                                gamma=None, grow_policy=None,
                                importance_type=None,
                                interaction_constraints=None,
learning_rate=None,
                                max_bin=None, max_cat_threshold=None,
                                max_cat_to_onehot=None,
max_delta_step=None,
                                max_depth=None, max_leaves=None,
                                min_child_weight=None, missing=nan,
                                monotone_constraints=None,
```

```
multi_strategy=None,
                                    n_estimators=None, n_jobs=None,
                                    num_parallel_tree=None, ...))])
```

REAL LIFE PREDICTION

```
new_person = pd.DataFrame([
    {
    "Income": 85000,
    "Ownership": "RENT",
    "Loan Amount": 15000,
    "Loan Intent ": "PERSONAL",
    "Interest Rate": 12.5,
    "Credit.Hist": 4,
    "Cr.Score": 690,
    "Previous Loans": "No"
    }
])

prediction = final_model.predict(new_person)
if prediction[0] == 1:
    print("Loan Approved")
else:
    print("Loan Rejected")
```

```
Loan Rejected
```