Business Problem/ Statement:

Loan approval is a critical process for financial institutions, and it often involves evaluating various factors such as credit history, income, employment status, and more. Building a predictive model for loan approval can help streamline this process, reduce manual effort, and make it more efficient while maintaining fairness and accuracy.

Scope of Work:

This project involves the development of a machine learning model with the objective of predicting the probability of loan approval. The model will be built using applicant information and historical loan data, aiming to deliver a solution that optimally benefits both lending institutions and loan applicants.

Project Overview:

This project endeavors to predict individual loan approval based on key features such as CIBIL Score, Education Level, and Annual Income.

Outline:

1. Data Collection
2. Feature Engineering
3. Data Cleaning
4. Exploratory Data Analysis
5. Data Preprocessing
6. Machine Learning Modeling with Hyperparameter Tuning
7. Model Evaluation

Dataset Features

Importing Packages

```
import pandas as pd
import numpy as np
import pickle
```

1. Data Collection

```
df = pd.read_csv("loan_approval_dataset.csv")
```

Get 5 rows of samples of the dataframe

```
df.sample(5)
```

```
      loan_id   no_of_dependents      education  self_employed
income_annum  \
593       594                  4       Graduate             No
600000
1502     1503                  2       Graduate            Yes
4800000
1174     1175                  4       Graduate            Yes
9800000
4098     4099                  0   Not Graduate            Yes
7300000
2833     2834                  0       Graduate            Yes
7500000

       loan_amount   loan_term   cibil_score
residential_assets_value  \
593        1800000           6           851
100000
1502      13200000          20           310
11800000
1174      27000000           2           395
18700000
4098      24000000          18           651
16600000
2833      28900000          10           632
21700000

       commercial_assets_value   luxury_assets_value
bank_asset_value  \
593                    1100000               2100000
800000
1502                   4600000              12900000
4200000
1174                   3000000              23000000
9700000
4098                    100000              19600000
4500000
2833                  10700000              24700000
9900000

      loan_status
593      Approved
1502     Rejected
1174     Rejected
4098     Approved
2833     Approved
```

2. Feature Engineering

Drop irrelavant columns

```
columns_to_remove = ['loan_id']

# Remove the specified columns
df.drop(columns=columns_to_remove, inplace=True)

# No. of rows & Columns in the dataframe (shape)
print("Dataset Shape:", df.shape)
```

```
Dataset Shape: (4269, 12)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0    no_of_dependents          4269 non-null   int64
 1    education                 4269 non-null   object
 2    self_employed             4269 non-null   object
 3    income_annum              4269 non-null   int64
 4    loan_amount               4269 non-null   int64
 5    loan_term                 4269 non-null   int64
 6    cibil_score               4269 non-null   int64
 7    residential_assets_value  4269 non-null   int64
 8    commercial_assets_value   4269 non-null   int64
 9    luxury_assets_value       4269 non-null   int64
 10   bank_asset_value          4269 non-null   int64
 11   loan_status               4269 non-null   object
dtypes: int64(9), object(3)
memory usage: 400.3+ KB
```

As we can see in the output.

- The dataset consists of `4269` records
- There are a total of `12` features (0 to 11)
- There are three types of datatype `dtypes:  int64(9), object(3)`
- It's Memory usage that is, `memory usage: 400.3+ KB`
- Also, We can check how many missing values available in the `Non-Null Count` column

```
# Descriptive statistics of the DataFrame
df.describe()
```

```
       no_of_dependents  income_annum   loan_amount    loan_term  \
count       4269.000000  4.269000e+03  4.269000e+03  4269.000000
mean           2.498712  5.059124e+06  1.513345e+07    10.900445
```

```
std            1.695910   2.806840e+06  9.043363e+06     5.709187
min            0.000000   2.000000e+05  3.000000e+05     2.000000
25%            1.000000   2.700000e+06  7.700000e+06     6.000000
50%            3.000000   5.100000e+06  1.450000e+07    10.000000
75%            4.000000   7.500000e+06  2.150000e+07    16.000000
max            5.000000   9.900000e+06  3.950000e+07    20.000000

        cibil_score   residential_assets_value
commercial_assets_value  \
count   4269.000000               4.269000e+03
4.269000e+03
mean     599.936051               7.472617e+06
4.973155e+06
std      172.430401               6.503637e+06
4.388966e+06
min      300.000000              -1.000000e+05
0.000000e+00
25%      453.000000               2.200000e+06
1.300000e+06
50%      600.000000               5.600000e+06
3.700000e+06
75%      748.000000               1.130000e+07
7.600000e+06
max      900.000000               2.910000e+07
1.940000e+07

        luxury_assets_value   bank_asset_value
count          4.269000e+03       4.269000e+03
mean           1.512631e+07       4.976692e+06
std            9.103754e+06       3.250185e+06
min            3.000000e+05       0.000000e+00
25%            7.500000e+06       2.300000e+06
50%            1.460000e+07       4.600000e+06
75%            2.170000e+07       7.100000e+06
max            3.920000e+07       1.470000e+07
```

Finding the unique values

```
def uniquevals(col):
    print(f'Unique Values in {col} is : {df[col].unique()}')

for col in df.columns:
    uniquevals(col)
    print("-"*75)

Unique Values in  no_of_dependents is : [2 0 3 5 4 1]
----------------------------------------------------------------------
-----
Unique Values in  education is : [' Graduate' ' Not Graduate']
```

```
----------------------------------------------------------------------
-----
Unique Values in  self_employed is : [' No' ' Yes']
----------------------------------------------------------------------
-----
Unique Values in  income_annum is : [9600000 4100000 9100000 8200000
9800000 4800000 8700000 5700000  800000
 1100000 2900000 6700000 5000000 1900000 4700000  500000 2700000
6300000
 5800000 6500000 4900000 3100000 2400000 7000000 9000000 8400000
1700000
 1600000 8000000 3600000 1500000 7800000 1400000 4200000 5500000
9500000
 7300000 3800000 5100000 4300000 9300000 7400000 8500000 8800000
3300000
 3900000 8300000 5600000 5300000 2600000  700000 3500000 9900000
3000000
 6800000 2000000 1000000  300000 6600000 9400000 4400000  400000
6200000
 9700000 7100000  600000 7200000  900000  200000 1800000 4600000
2200000
 2500000 8600000 4000000 5200000 8900000 1300000 4500000 8100000
9200000
 2800000 7500000 6400000 6900000 7700000 3200000 7900000 5900000
3400000
 2100000 3700000 5400000 2300000 7600000 6000000 6100000 1200000]
----------------------------------------------------------------------
-----
Unique Values in  loan_amount is : [29900000 12200000 29700000
30700000 24200000 13500000 33000000 15000000
  2200000  4300000 11200000 22700000 11600000 31500000  7400000
10700000
  1600000  9400000 10300000 14600000 19400000 14000000 25700000
1400000
  9800000  9500000 28100000  5600000 24000000 25300000 12000000
22000000
 11900000  3400000  6200000 27200000  7700000  5100000 18100000
24900000
  2300000 13400000 27800000 19100000 20500000 25400000 24700000
7600000
 23000000 19700000 24500000 10600000 30500000 18400000 18200000
18900000
 28900000  7500000 12300000 29100000 10100000 12400000  5000000
1500000
 18600000 18300000 16700000  8400000  6500000 14800000 33500000
29400000
  8900000 31200000 21200000  8600000  8200000  3800000 28300000
8000000
 37600000 21100000 20700000  6400000  2000000  1100000 25000000
```

```
10800000
   900000 12900000  4500000 23600000  9700000 35900000  6800000
22100000
 23400000 23200000 15800000 32900000  3200000 18700000 19500000
600000
   800000  2600000  1200000 20800000 22600000  3600000 13900000
5500000
  6700000  8500000   700000 17400000 32100000 11100000 19300000
28800000
 20600000 35000000 33300000  1300000  9600000 15100000  5300000
22300000
 15900000 12800000 35200000 17500000 10500000  4100000 28200000
14300000
 13300000 17900000  9900000 23100000  3100000 10900000 30400000
23300000
 19800000  2900000 13200000 27100000  6000000 16400000 15600000
30100000
 20900000 15400000  3300000 32700000 15200000  7800000 17000000
11300000
 10400000 11000000  1700000 27000000  3500000 32400000 34600000
15500000
 22500000 16200000 29300000  9100000 30900000  4700000  2400000
35400000
 20000000 38800000  8100000 19600000 34300000 22200000 14400000
16800000
 27900000 20400000  4900000  4000000 19900000  1800000 11800000
25500000
  9300000 20200000  1000000 38200000  6600000 33200000 24400000
14100000
 28700000 23500000 17100000  5700000  7000000 16900000 21000000
12600000
 28000000 17200000 24100000 26300000 38400000 32200000 26900000
21900000
 12500000 29800000 17600000 29000000 35300000 14500000 22400000
8700000
  6300000  5200000  2700000 25900000 21500000 15700000  7300000
7200000
 18800000  2100000  4600000  5400000 15300000  2800000 36400000
27500000
 11400000 18500000 26700000  5800000 21400000  2500000 14700000
17300000
 28600000 31900000  8800000 30300000 24800000 19200000 39500000
9000000
 19000000 23900000 31800000 10000000 14200000 22800000 32300000
500000
  6100000  5900000 24600000 16000000 12700000 13100000  6900000
27700000
 29200000 13800000 18000000 27400000 17800000 20100000 32600000
4800000
```

```
 11500000 22900000  1900000   300000 16600000 28500000 25100000
25800000
 21800000 30000000 10200000 31700000 26000000  8300000 35500000
33900000
 17700000  4200000  9200000 20300000   400000 34700000 26500000
16500000
 14900000 37000000 27300000 26200000 25600000 16300000 24300000
21600000
 11700000 34200000 34500000 13000000 23700000 30800000  3900000
13600000
 38700000 26600000 37900000 21700000 29600000 23800000 34000000
25200000
 35700000 26100000 16100000 13700000 38000000 37500000  7900000
34400000
 37300000 21300000 28400000 35800000 38500000 34900000 33600000
36800000
 31400000  3000000  4400000 26400000 37800000  7100000 34100000
30200000
 32000000 31300000 12100000 36700000 30600000  3700000 31600000
29500000
 31000000 34800000 36500000 36000000 36300000 31100000 26800000
35100000
 32800000 33100000 32500000 33400000 27600000 33700000 36600000
33800000
 37700000 36100000]
----------------------------------------------------------------------
-----
Unique Values in  loan_term is : [12  8 20 10  4  2 18 16 14  6]
----------------------------------------------------------------------
-----
Unique Values in  cibil_score is : [778 417 506 467 382 319 678 782
388 547 538 311 679 469 794 663 780 736
 652 315 530 551 324 514 696 662 336 850 313 363 436 830 612 691 636
348
 352 712 822 540 342 787 331 677 634 502 435 689 657 590 818 431 841
421
 797 478 669 365 586 784 364 715 693 777 312 340 386 418 735 494 671
697
 801 576 639 470 826 613 713 439 387 402 837 641 489 844 452 366 300
861
 562 463 702 618 633 764 591 719 317 302 879 437 456 647 379 717 545
570
 865 821 859 395 429 565 357 465 479 425 786 564 501 727 894 829 802
543
 772 572 709 481 306 415 548 701 890 704 318 761 524 681 737 638 656
341
 371 886 748 376 873 309 869 534 566 742 824 575 766 888 622 458 327
682
 583 816 455 355 389 870 827 768 707 665 420 471 819 809 744 484 673
```

```
695
 473 491 733 434 774 503 598 796 632 770 667 585 851 378 807 831 674
725
 600 536 477 560 539 852 853 729 546 789 325 716 523 345 649 666 813
599
 513 483 308 651 433 403 405 516 468 672 549 450 320 476 573 877 531
474
 499 726 485 708 404 512 441 555 466 427 593 731 451 628 424 381 449
445
 781 721 563 419 372 885 596 349 685 377 620 611 767 592 900 814 755
584
 380 655 833 658 648 730 621 610 339 650 367 847 360 880 608 760 385
710
 711 855 771 338 769 629 699 391 891 775 897 839 868 353 792 635 457
350
 874 411 482 396 303 728 698 490 504 790 860 492 834 443 329 739 867
307
 375 601 756 838 442 808 597 373 552 607 823 328 580 559 587 817 765
383
 843 783 409 625 645 887 791 686 722 407 895 453 627 889 684 578 369
557
 519 741 508 493 664 362 703 758 828 528 623 579 846 589 845 401 522
588
 863 798 668 881 406 799 743 734 812 459 448 517 426 785 472 683 803
361
 464 747 335 394 848 788 509 899 595 322 631 330 567 323 670 609 354
746
 857 556 393 688 384 414 815 854 849 346 856 440 616 461 866 820 544
561
 614 351 399 344 301 763 624 644 642 423 724 706 811 326 488 475 337
511
 810 428 356 594 480 757 321 368 806 832 571 527 333 532 754 835 553
400
 558 515 740 447 745 495 660 883 795 762 462 779 752 305 310 525 661
884
 800 568 842 653 617 460 804 358 836 554 840 430 347 550 878 603 444
875
 343 714 529 446 705 898 487 615 676 605 569 410 753 454 619 858 392
637
 359 723 304 690 862 496 659 542 749 694 574 692 521 541 640 630 535
422
 606 370 700 751 896 577 537 316 412 793 390 397 876 498 872 871 497
759
 413 602 720 505 582 416 510 500 626 654 892 680 750 314 520 776 825
646
 518 805 332 882 604 507 408 374 687 533 581 675 773 718 432 526 398
643
 893 438 486 732 334 738 864]
----------------------------------------------------------------------
```

```
-----
Unique Values in  residential_assets_value is : [ 2400000  2700000
7100000 18200000 12400000  6800000 22500000 13200000
  1300000  3200000  8100000 15300000  6400000 10800000  1900000
5700000
  2900000  1000000 10300000  9500000  3800000 13100000   900000
7900000
 11500000  4500000  2300000 21800000 20200000  3600000   700000
9700000
  3400000  7000000   100000  8600000 22300000   200000  2200000
13000000
  5400000   800000   500000  8700000 15400000  7400000  1200000
2100000
 19300000 18500000  -100000 23800000  4700000 24400000  1600000
7600000
  6100000  5500000  4000000 18400000  3900000  6500000   600000
14300000
 11600000 17600000 25500000  9400000  5300000 17100000 20400000
5100000
 24100000 19200000  9100000 14700000 25900000   300000 11400000
7800000
 19600000  5600000  7300000 19500000 16100000  1500000 12700000
26800000
 12200000   400000 15100000 11700000  4400000  6600000  1100000
2600000
 14600000 13600000 15900000        0  5800000  3700000 24200000
4900000
  2500000  7700000 21900000  3300000  9800000 12100000  3000000
16800000
 12600000  1700000  8800000 13700000 10000000  6300000 15200000
22000000
  8300000 11300000 14400000 11100000  3100000 15500000  3500000
13800000
  9000000 14100000 14800000  8500000 18700000  2800000  9200000
20000000
  4100000 22800000 16500000  6000000 23200000  5000000 25600000
24500000
 13400000 14000000 16000000 18100000  8000000  9900000 17200000
1800000
  1400000 10400000  4200000  6900000 16600000  9600000 17400000
8400000
 11900000 10500000  5900000  7200000 14200000 22900000  4300000
16900000
  6200000 12500000 15700000  9300000 18000000  8200000 10700000
4800000
 10200000 21500000 12900000  4600000 15600000 10600000  5200000
21700000
 11000000 23300000 20800000 23000000 11800000 21100000 10900000
2000000
```

```
 15800000 23400000 13500000 23900000 17300000 18300000 19400000
22100000
 12000000 24000000  6700000 13900000 20600000 25400000  7500000
10100000
 17700000 28300000 11200000 18800000 14500000 24900000 26300000
13300000
 22400000 27600000 21400000 28700000 25300000 25800000 18600000
19100000
 22200000 28200000 19700000 25200000 24700000 16700000 17000000
16300000
 15000000 21300000 12800000 20300000 12300000 19900000 16200000
19000000
 16400000  8900000 22700000 25700000 21200000 27000000 21600000
17800000
 28500000 14900000 17900000 28400000 23700000 20500000 24600000
20100000
 22600000 20900000 21000000 26600000 26200000 19800000 17500000
28000000
 24800000 26900000 26100000 20700000 29100000 18900000 25100000
23500000
 24300000 27500000 25000000 23100000 27400000 27300000]
----------------------------------------------------------------------
-----
Unique Values in  commercial_assets_value is : [17600000  2200000
4500000  3300000  8200000  8300000 14800000  5700000
   800000  1400000  4700000  5800000  9600000 16600000  1200000
3900000
   100000  2800000        0  3500000  1600000 11300000  1700000
600000
  8700000  3100000 10600000  4200000 11900000 12400000  5200000
7400000
   200000   700000   300000  1300000 11200000 12100000  1500000
6300000
  6900000  9100000  8600000 10500000  1800000  9300000  5600000
10300000
  4900000 16300000  1900000  6100000  9700000 11700000  9400000
3800000
  2500000  7800000  8900000   500000 11400000 13600000  2600000
4300000
  3200000  1100000   400000  4800000  8500000 15200000  3600000
16500000
  2700000  7600000  6000000 12200000  2000000  1000000  6200000
8000000
  5900000  4100000  6500000 10000000 16700000   900000  2100000
9500000
  5500000  4400000 18700000  5100000 11100000 12600000  5000000
6800000
  2400000  7500000  2900000 10900000 11000000 11600000  2300000
3400000
```

```
 11500000  8100000  5300000  6700000 10200000 10800000  4000000
4600000
  7000000  6600000 17500000 16200000 12300000 12800000 13200000
16400000
 19000000 16100000  8800000  3700000  5400000  8400000 12000000
15000000
  9200000 17200000 11800000 14900000 13800000  7900000 10400000
18500000
 12500000 13400000  9900000 12700000 15400000 14700000 15600000
14000000
 16000000 13000000 14300000  9800000 18800000 13900000  7200000
7100000
 15100000 15500000 13300000  3000000 13700000  7300000 17800000
6400000
 17900000 12900000 14600000 10100000 18300000  9000000 14500000
14200000
 17300000 13100000 10700000 16800000 18900000 18400000 18200000
14100000
 14400000  7700000 17000000 15900000 15300000 19400000 16900000
13500000
 17400000 15700000 15800000 17700000]
----------------------------------------------------------------------
-----
Unique Values in  luxury_assets_value is : [22700000  8800000 33300000
23300000 29400000 13700000 29200000 11800000
  2800000  3300000  9500000 20400000 14600000 20900000  5900000
16400000
  1300000  6700000  6200000 23500000 18000000 22200000 19500000
1100000
 10000000  6600000 25300000  5400000 27500000 33700000 25500000
21700000
  2200000 19900000 19000000  6000000  5300000 16700000  5600000
31000000
  3900000  1800000 16200000 21400000  8700000 17700000 18500000
37700000
 20500000 21800000  9300000 31900000 19400000 16300000 34600000
17500000
 18600000 25900000 26500000 27400000 10500000 13100000 14900000
24100000
  4900000  1900000 11900000 21500000 12600000  4800000 12900000
35400000
 25200000  2400000 12300000 26600000 10300000 11000000  3800000
27900000
 23400000 12500000 22400000  3200000   700000 18200000 23200000
36400000
 13800000  1200000   500000 11400000  4100000 23800000 20800000
9900000
 11700000   900000 17900000 19300000 33400000  7700000 22600000
1500000
```

```
 23600000  2700000  2000000   800000 15500000 33900000 25700000
4400000
 13900000  8600000  7500000  7400000 12800000 24500000  2100000
5100000
  7200000 18800000 18100000  2900000 36100000 14000000  8400000
27800000
  8000000 10400000 17800000  4300000 27000000 16000000  5000000
23100000
 18700000  9700000 17400000  6500000 33800000  5700000 20000000
8200000
 14300000 26300000 26900000 26400000 27700000 24000000 22500000
28000000
 31800000 12200000 38200000 38600000 19600000 21900000  3500000
27200000
  3700000 15000000 34700000 23700000  8500000 10600000 16100000
21200000
 13600000  7000000 18400000  7100000 14700000  9600000 11200000
24300000
 20300000  6400000 23000000 25000000  6300000 22800000 31600000
29700000
 29100000 30800000 13000000 26000000 14500000 16800000 29500000
28200000
 19100000 26700000  5200000  4600000  7900000 16900000  9800000
15600000
 30500000 30200000 12400000  3000000 20100000  8900000 19800000
35500000
 28500000 25400000 16500000 17200000  4700000 28800000  3600000
400000
 14200000 14800000 14100000 22900000 26100000 36500000 28600000
34500000
 30900000  6900000  2300000 25100000 28900000 14400000 29900000
7600000
 37000000 15800000  1400000 33500000 13500000   300000 13200000
1600000
  8100000  5800000 10900000 11300000 10200000 13300000 34900000
17300000
 22000000 32100000 20700000 26800000 27100000 10100000 21000000
19700000
   600000  7300000 32000000 22100000  2600000  9100000 31100000
32700000
 32800000 24900000  5500000 32600000  3400000  9000000 12700000
6800000
 17100000 20200000 10800000 34100000 26200000 29000000 11600000
31300000
 28400000 11100000 12000000 12100000 17000000 15100000 28300000
16600000
 15300000 18900000 23900000 24400000 17600000 11500000 21100000
30000000
 29600000 15200000 27600000 20600000 30400000  9400000  7800000
```

```
18300000
  4200000  8300000 30100000 25800000  1700000 21600000 29300000
35700000
  4500000 30300000 10700000 24800000 31500000 24700000 19200000
13400000
 35100000 35600000 15900000 33000000 31700000  9200000  6100000
15400000
  2500000 24600000 35800000 22300000 34300000 36600000  3100000
28700000
 36900000 28100000 32500000 38100000 39200000 15700000 37800000
27300000
 31200000 39100000 21300000 24200000 37200000 37900000 25600000
33600000
 30600000 32900000 37400000 34000000  1000000 37600000 35900000
32300000
 32400000 31400000 30700000 34400000  4000000 32200000 29800000
33200000
 34800000 36000000 36200000 36800000 35300000 38000000 33100000
37300000
 34200000 35000000 36700000]
----------------------------------------------------------------------
-----
Unique Values in  bank_asset_value is : [ 8000000  3300000 12800000
7900000  5000000  5100000  4300000  6000000
   600000  1600000  3100000  6400000  1900000  4400000   700000
5900000
  6100000  5400000  8500000   300000  2600000  7200000  2500000
9700000
  9300000  1000000  5800000   900000  1400000  7100000  2900000
9000000
  5200000   800000 10900000  4900000  6500000  8200000 11700000
10500000
 11300000  3400000  6200000  8700000  4100000  4800000 11400000
4700000
  2800000 11900000  5500000  2400000  4200000  7600000  5600000
2000000
  1100000  6300000 11100000  8600000  6800000  3600000 10200000
12700000
  2100000  1300000   400000  7000000  7300000   100000   200000
11600000
  1800000  9800000  8100000  7500000 13400000  9600000  3800000
8400000
  3200000  1200000  4600000  8300000  4500000  3500000  2300000
7400000
  1700000  9500000  3000000  2200000  9200000  4000000 11200000
500000
  9400000 14400000 10000000  6600000 12500000  1500000  9100000
7700000
  7800000 10300000  9900000  8800000  5700000 10400000 11800000
```

```
5300000
 12400000  2700000 11500000  3900000        0 10800000  6700000
12900000
 12300000  6900000 12200000 13500000  8900000  3700000 12100000
13600000
 13100000 10600000 13900000 12000000 13000000 10100000 10700000
11000000
 13200000 14700000 14000000 13300000 13800000 14600000 14300000
14200000
 13700000 14100000]
----------------------------------------------------------------------
-----
Unique Values in  loan_status is : [' Approved' ' Rejected']
----------------------------------------------------------------------
-----
```

Checking categorical and numerical features

```
# Select all categorical data type and stored in one dataframe and
select all other categorical and stored in one data frame
cat_var = df.select_dtypes(include=['object']).columns
num_var = df.select_dtypes(include =
['int32','int64','float32','float64']).columns

cat_var ,num_var

(Index([' education', ' self_employed', ' loan_status'],
dtype='object'),
 Index([' no_of_dependents', ' income_annum', ' loan_amount', '
loan_term',
        ' cibil_score', ' residential_assets_value', '
commercial_assets_value',
        ' luxury_assets_value', ' bank_asset_value'],
       dtype='object'))
```

3. Data Cleaning

Checking for duplicate rows

```
# Check for duplicates based on all columns
duplicates_all = df[df.duplicated()]

# Print the results
print("Duplicates based on all columns:")
print(duplicates_all)

Duplicates based on all columns:
Empty DataFrame
```

```
Columns: [ no_of_dependents,  education,  self_employed,
income_annum,  loan_amount,  loan_term,  cibil_score,
residential_assets_value,  commercial_assets_value,
luxury_assets_value,  bank_asset_value,  loan_status]
Index: []
```

It is clear that there are no duplicates

Handling null values

```
df.isna().sum()

 no_of_dependents            0
 education                   0
 self_employed               0
 income_annum                0
 loan_amount                 0
 loan_term                   0
 cibil_score                 0
 residential_assets_value    0
 commercial_assets_value     0
 luxury_assets_value         0
 bank_asset_value            0
 loan_status                 0
dtype: int64
```

Counting zeros of each column

```
# Count zeros in each column
zero_counts = (df == 0).sum()

print(zero_counts)

 no_of_dependents            712
 education                     0
 self_employed                 0
 income_annum                  0
 loan_amount                   0
 loan_term                     0
 cibil_score                   0
 residential_assets_value     45
 commercial_assets_value     107
 luxury_assets_value           0
 bank_asset_value              8
 loan_status                   0
dtype: int64
```

We are assuming residential_assets_value, commercial_assets_value and bank_asset_value to be non zero therefore replacing zeros with mean. But we do not consider no of dependents

```python
# Calculate means for the columns
mean_residential_assets = df[' residential_assets_value'].mean()
mean_commercial_assets = df[' commercial_assets_value'].mean()
mean_bank_assets = df[' bank_asset_value'].mean()

# Replace zeros with means in the specified columns
df[' residential_assets_value'].replace(0, mean_residential_assets,
inplace=True)
df[' commercial_assets_value'].replace(0, mean_commercial_assets,
inplace=True)
df[' bank_asset_value'].replace(0, mean_bank_assets, inplace=True)
```

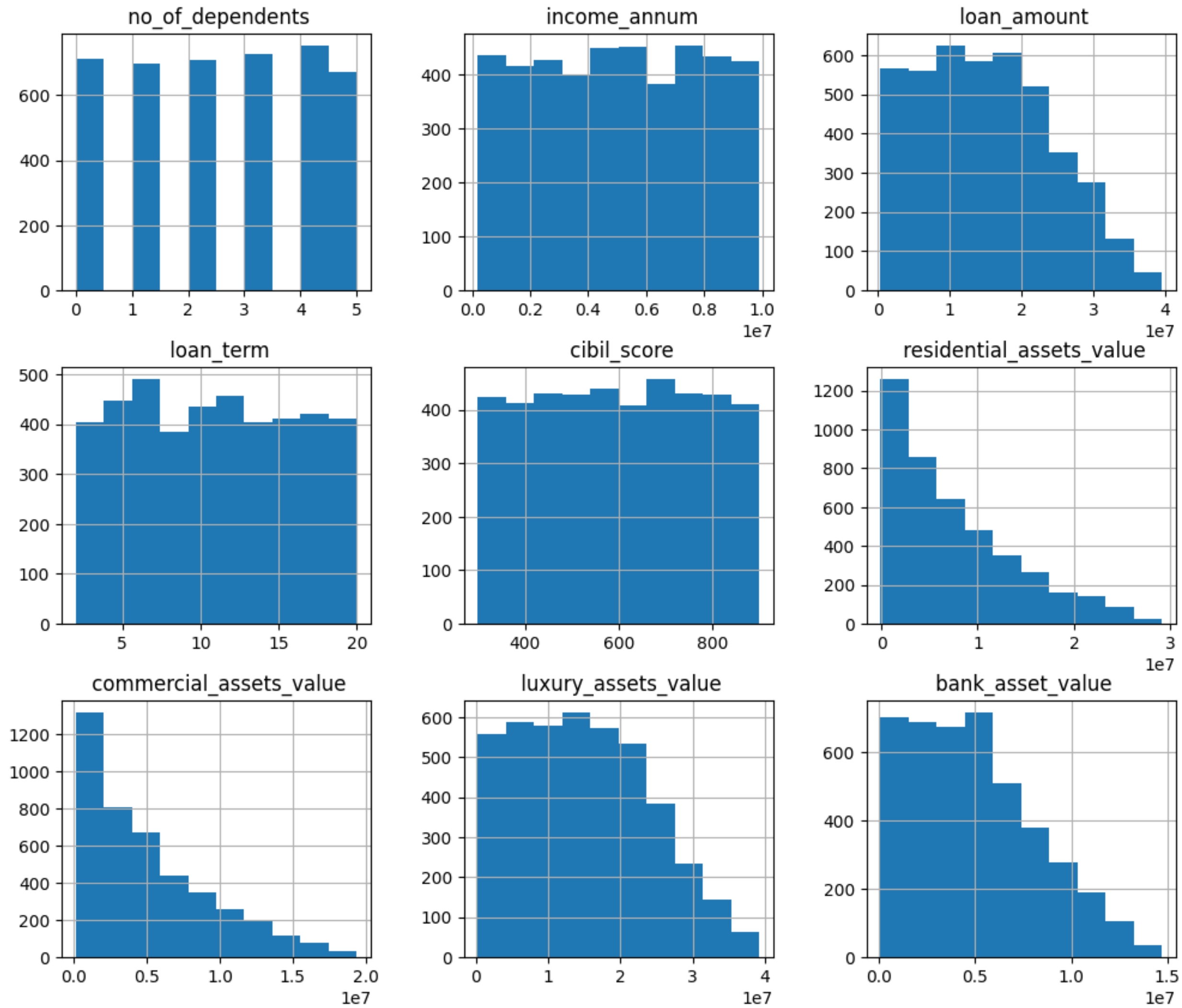
Now, 'df' contains the zeros in the specified columns replaced with their means

```python
(df == 0).sum()
```

```
 no_of_dependents            712
 education                     0
 self_employed                 0
 income_annum                  0
 loan_amount                   0
 loan_term                     0
 cibil_score                   0
 residential_assets_value      0
 commercial_assets_value       0
 luxury_assets_value           0
 bank_asset_value              0
 loan_status                   0
dtype: int64
```

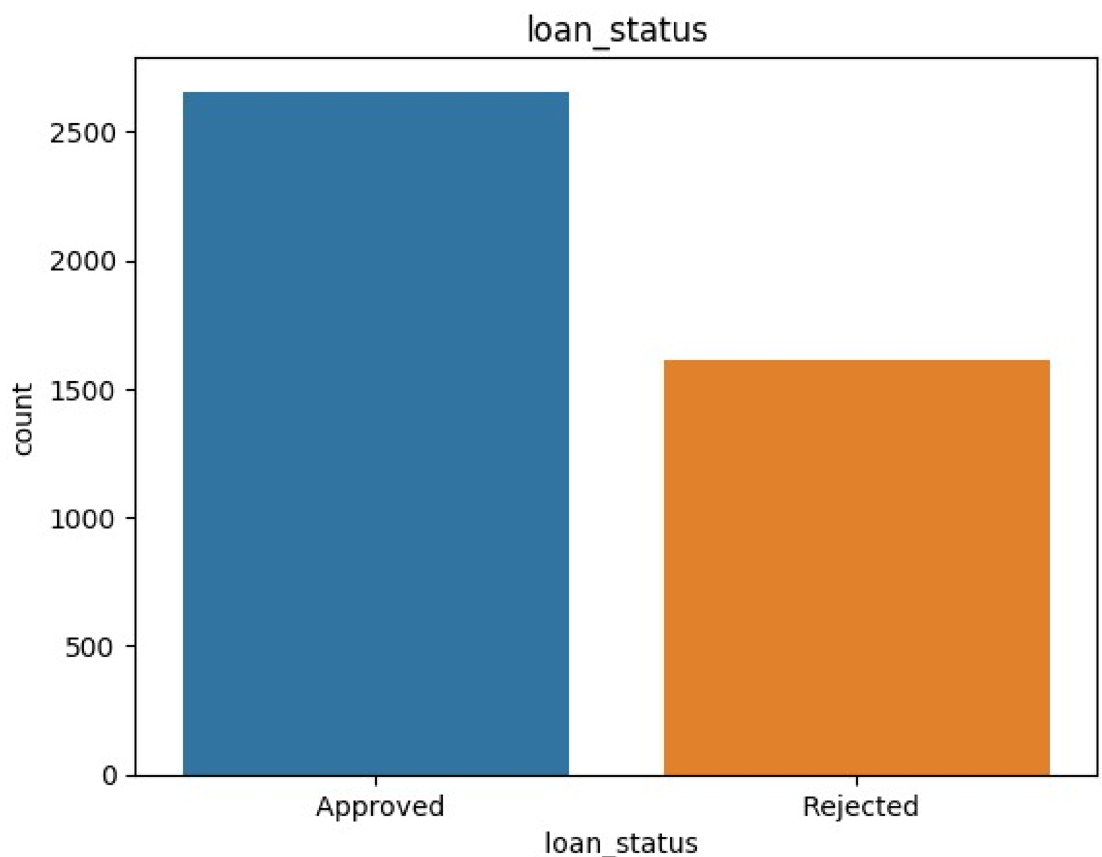
4. Exploratory Data Analysis

```python
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

import warnings
# Ignore FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)

plot = df.hist(figsize=(12,10))
```



Loan Status Distribution

```
sns.countplot(x = ' loan_status', data = df).set_title('loan_status')

Text(0.5, 1.0, 'loan_status')
```

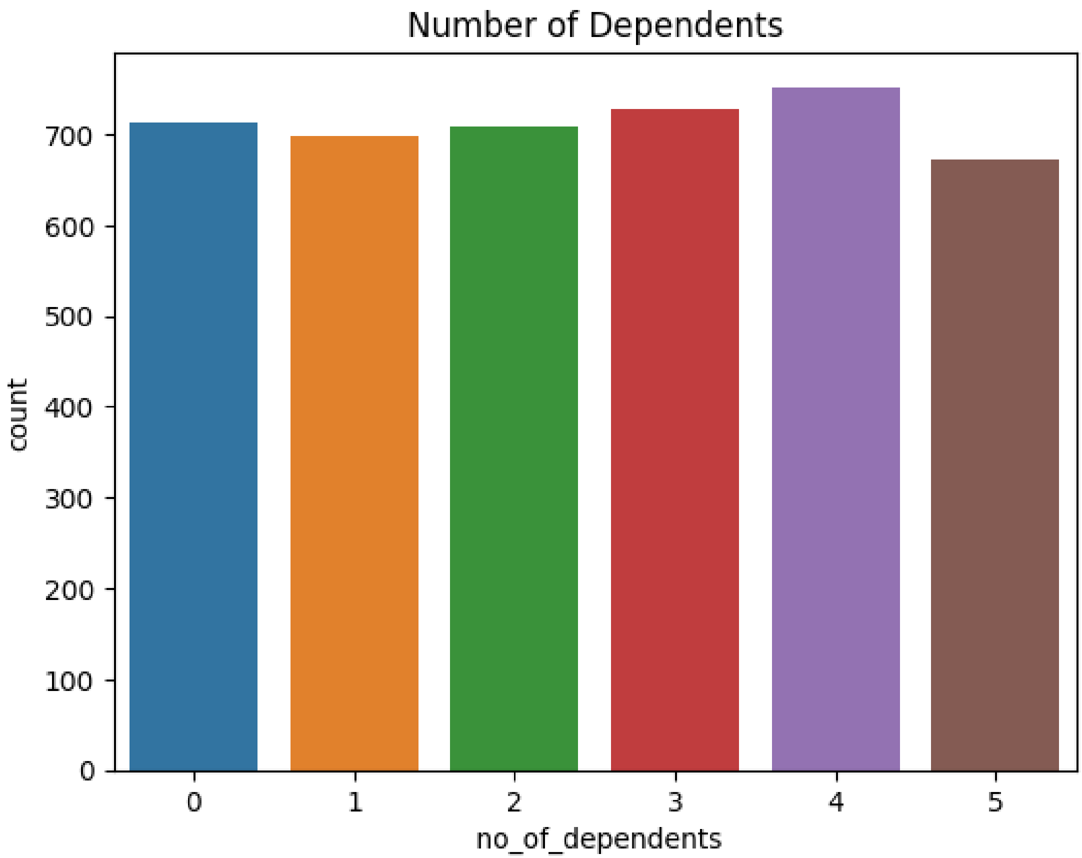


Dataset is clearly unbalanced (Approved > Rejected)

Number Of Dependents Distribution

```
sns.countplot(x = ' no_of_dependents', data = df).set_title('Number of Dependents')
```

```
Text(0.5, 1.0, 'Number of Dependents')
```

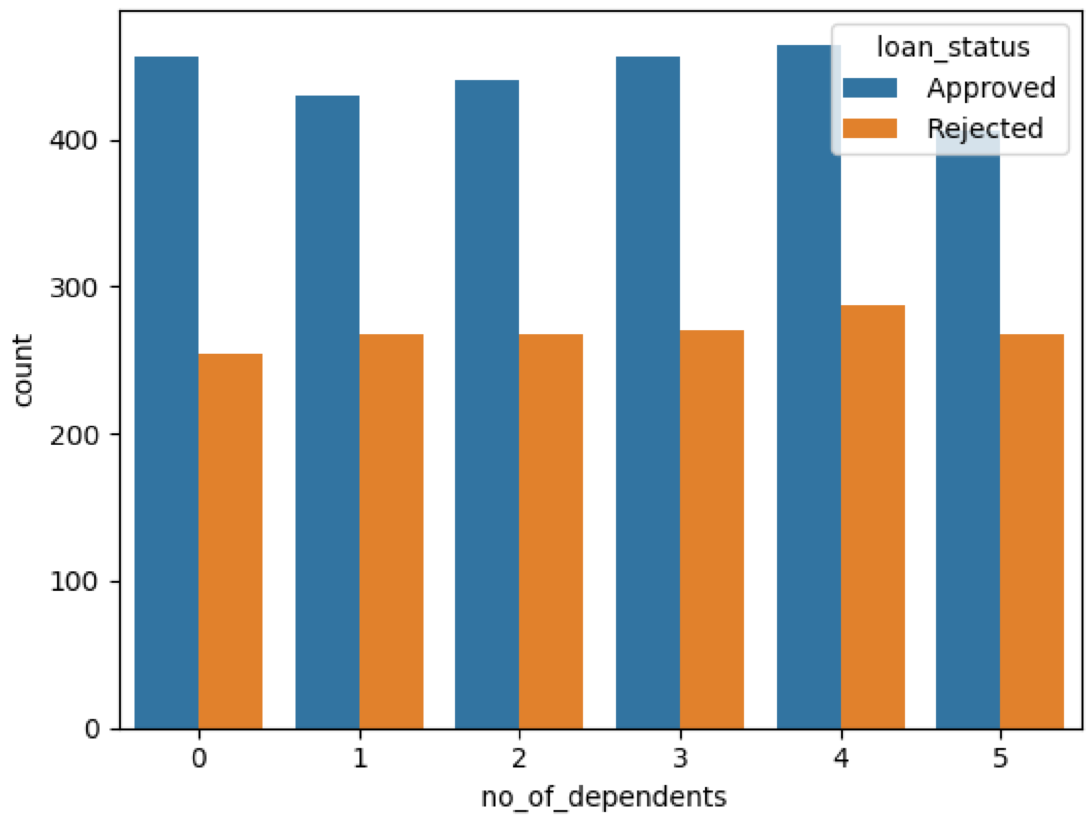


The graph illustrates the number of dependent individuals associated with loan applicants, revealing a stark contrast in living arrangements. There is not much difference in the number of dependents,Since the number of dependents increases the disposable income of the applicant decreases. So I assume that that the number of applicants with 0 or 1 dependent will have higher chances of loan approval.

Number of Dependants Vs Loan Status

```
sns.countplot(x = ' no_of_dependents', data = df, hue = '
loan_status')
```

```
<Axes: xlabel=' no_of_dependents', ylabel='count'>
```

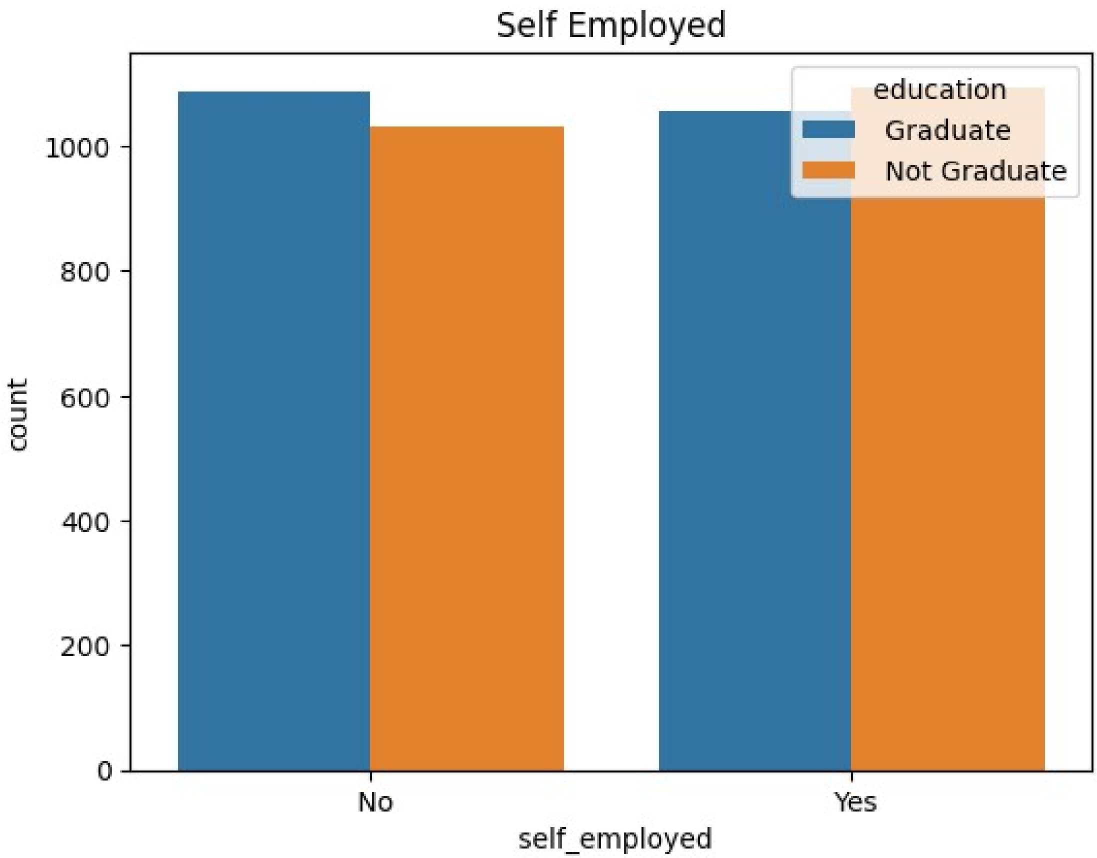


The graph tells us that when someone has more family members they take care of, their chances of loan rejection go up. But what's interesting is that the number of people who get loans approved doesn't change much, even if they have more family members. This means my guess that loans might be approved less often for people with more family members isn't really right, based on this graph. It shows that sometimes what we think might not match what actually happens.

Education and Self Employed

```
sns.countplot(x=' self_employed', data = df, hue = '
education').set_title('Self Employed')
```

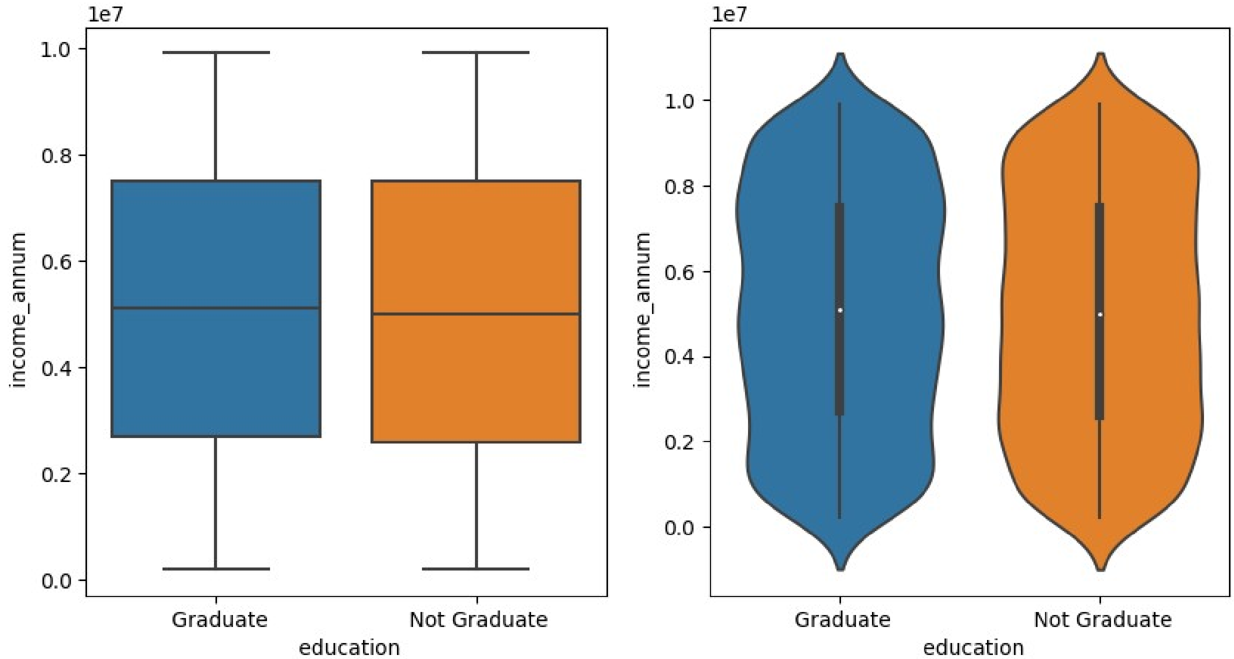
```
Text(0.5, 1.0, 'Self Employed')
```



The graph depicting the relationship between the employment status of applicants and their education levels highlights important trends for loan approval considerations. It reveals that a majority of non-graduate applicants are self-employed, while most graduate applicants are not self-employed. This indicates that graduates are more likely to be employed in salaried positions, whereas non-graduates tend to be self-employed.

Education and Income

```
fig, ax = plt.subplots(1,2,figsize=(10, 5))
sns.boxplot(x = ' education', y = ' income_annum', data = df,
ax=ax[0])
sns.violinplot(x = ' education', y = ' income_annum', data = df,
ax=ax[1])

<Axes: xlabel=' education', ylabel=' income_annum'>
```



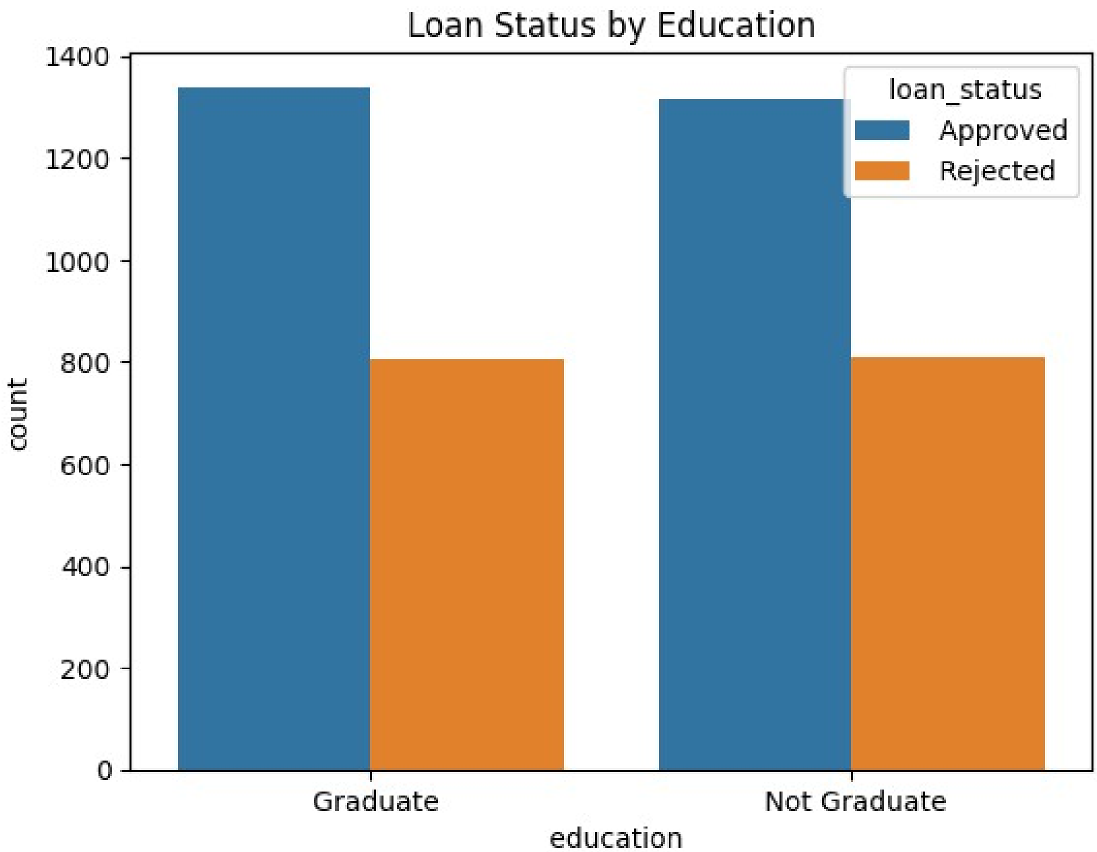
The combination of boxplot and violinplot visualizations provides insights into the relationship between education levels of loan applicants and their annual incomes. The boxplot reveals that both graduates and non-graduates have similar median incomes, indicating that having a degree doesn't necessarily lead to a significant income advantage.

Moreover the violinplot shows the distribution of income among the graduates and non graduate applicants, where we can see that non graduate applicants have a even distribution between income `2000000 and 8000000` , whereas there is a uneven distribution among the graduates with more applicants having income between `6000000` and `8000000` Since there is not much change in annual income of graduates and non graduates, I assume that education does not play a major role in the approval of loan.**

Education Vs Loan Status

```
sns.countplot(x = ' education', hue = ' loan_status', data =
df).set_title('Loan Status by Education')
```

```
Text(0.5, 1.0, 'Loan Status by Education')
```

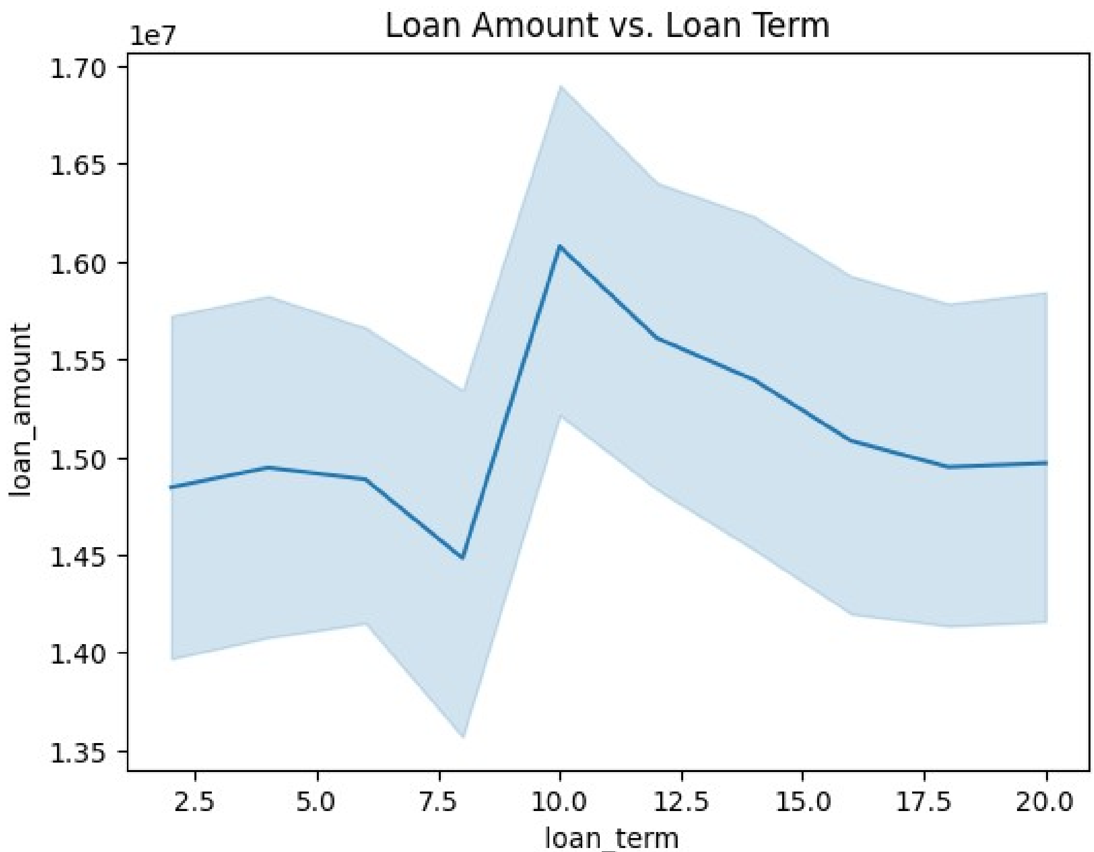


The graph indicates that there's only a small difference between the number of loans approved and rejected for both graduate and non-graduate applicants. This difference is so small that it doesn't seem to be significant.

Loan Amount vs Terms

```
sns.lineplot(x = ' loan_term', y = ' loan_amount', data =
df).set_title('Loan Amount vs. Loan Term')
```

```
Text(0.5, 1.0, 'Loan Amount vs. Loan Term')
```




```
df.head()

   no_of_dependents      education  self_employed   income_annum  \
0                 2       Graduate             No        9600000
1                 0   Not Graduate            Yes        4100000
2                 3       Graduate             No        9100000
3                 3       Graduate             No        8200000
4                 5   Not Graduate            Yes        9800000

   loan_amount   loan_term   cibil_score
residential_assets_value  \
0      29900000          12           778                   2400000.0

1      12200000           8           417                   2700000.0

2      29700000          20           506                   7100000.0

3      30700000           8           467                  18200000.0

4      24200000          20           382                  12400000.0


   commercial_assets_value   luxury_assets_value
bank_asset_value  \
```

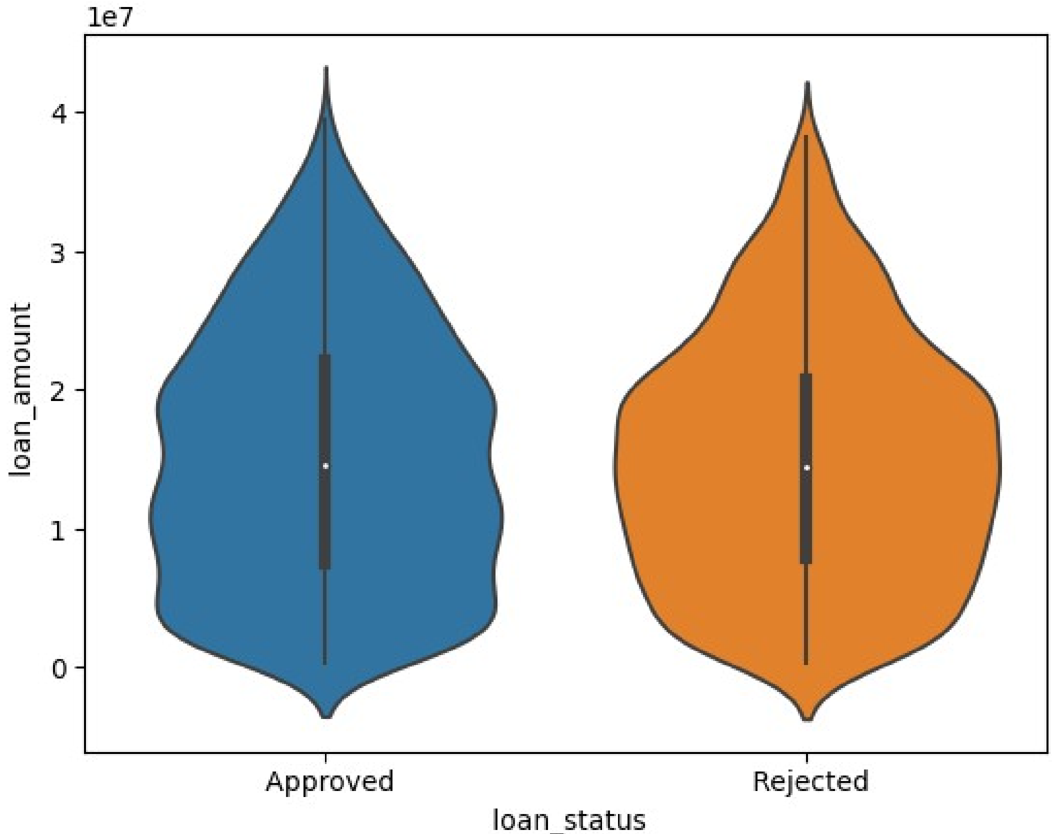
```
0          17600000.0          22700000          8000000.0

1           2200000.0           8800000          3300000.0

2           4500000.0          33300000         12800000.0

3           3300000.0          23300000          7900000.0

4           8200000.0          29400000          5000000.0


   loan_status
0     Approved
1     Rejected
2     Rejected
3     Rejected
4     Rejected
```

This line plot shows the trend between the loan amount and the loan tenure. Between the loan tenure of `2.5 - 7.5`years the loan amount is between `1400000 - 15500000`.

However the loan amount is significantly higher for the loan tenure of `10 years`. There is a huge difference

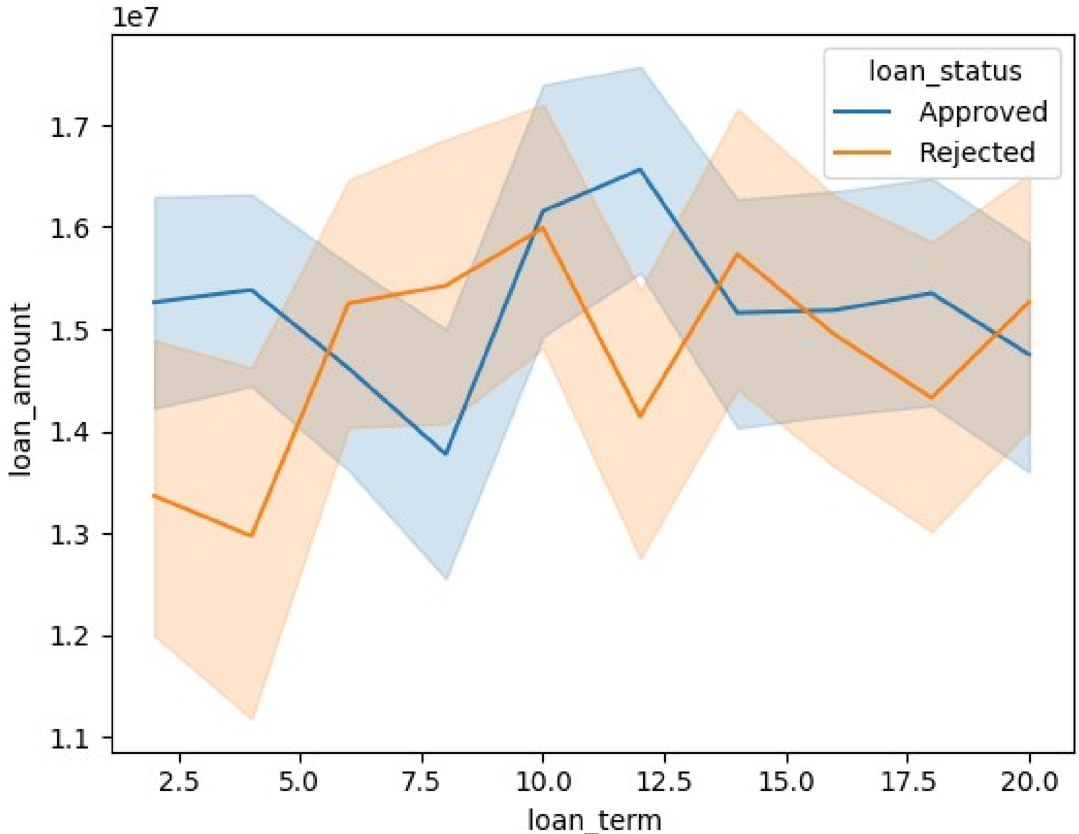
Loan Amount vs Loan Status

```
sns.violinplot(x=' loan_status', y=' loan_amount', data=df)

<Axes: xlabel=' loan_status', ylabel=' loan_amount'>
```



Loan Amount & Tenure vs Loan Status

```
sns.lineplot(x=' loan_term', y=' loan_amount', data=df, hue='
loan_status')

<Axes: xlabel=' loan_term', ylabel=' loan_amount'>
```



The graph shows how loan amount, the time to repay, and loan approval are connected. It's clear that loans that are accepted often have higher amounts and shorter repayment times. On the other hand, loans that are rejected are usually for lower amounts and longer repayment periods. This could be because the bank prefers to approve loans that are easier to pay back quickly and that bring in more profit. They might not want to deal with very small loans due to the costs involved. However, other things like how reliable the person borrowing is with money also matter in these decisions. The graph gives us a glimpse into how banks think when they decide to approve or reject loans.

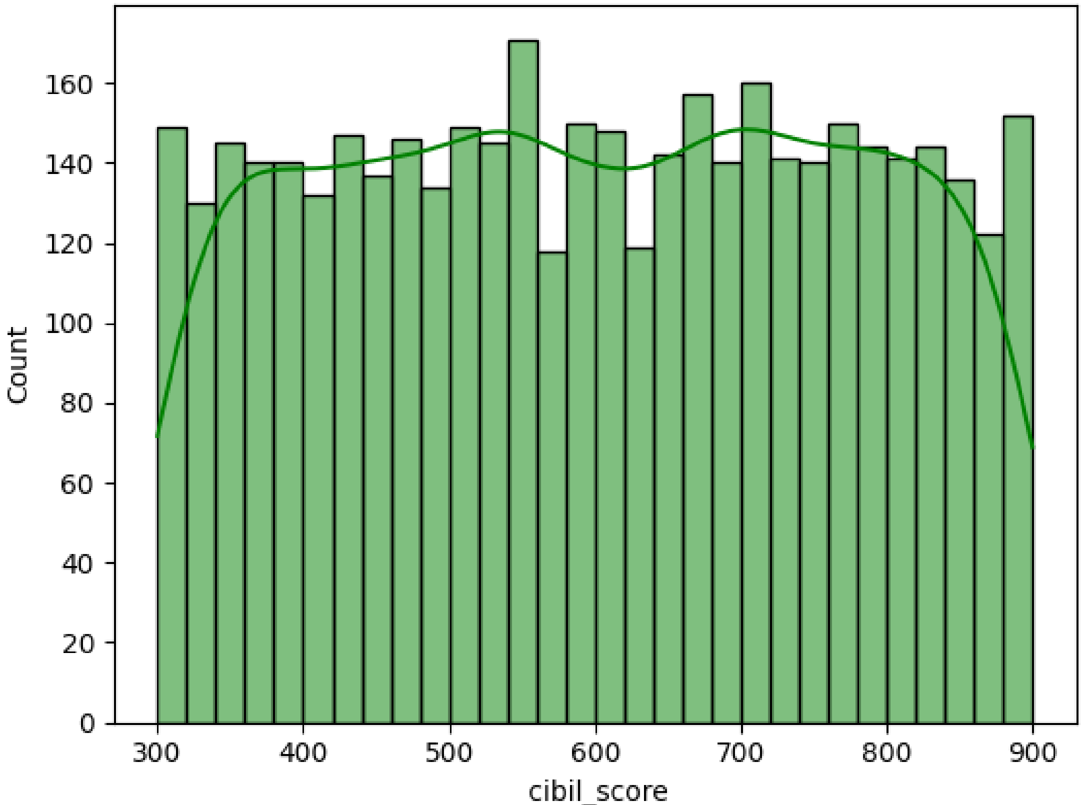
CIBIL Score Distribution

CIBIL Score ranges and their meaning.

\begin{array}{|c|c|} \hline CIBIL & Meaning \ \hline 300-549 & Poor \ 550-649 & Fair \ 650-749 & Good \ 750-799 & Very Good \ 800-900 & Excellent \ \hline \end{array}

```python
# Viewing the distribution of the cibil_score column
sns.histplot(df[" cibil_score"], bins=30, kde=True, color='green')
```

```
<Axes: xlabel=' cibil_score', ylabel='Count'>
```

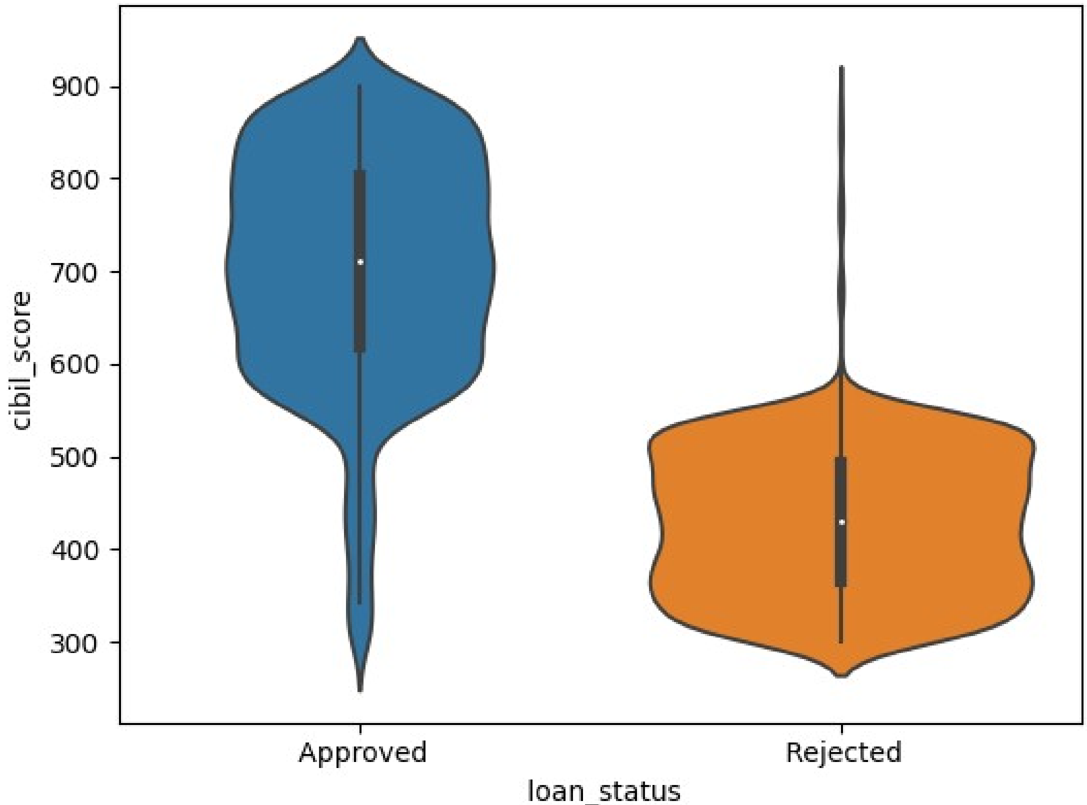


Looking at the table, most customers have low CIBIL scores (below 649), which could make it hard for them to get loans approved. But there's a good number of customers with high scores (above 649), which is positive for the bank. The bank can give these high-score customers special treatment like good deals and offers to get them interested in taking loans from the bank. Based on this, we can guess that people with high CIBIL scores are more likely to get their loans approved. This is because higher scores usually mean they are better with money. Overall, the bank can use this information to make decisions that help both the bank and its customers.

CIBIL Score Vs Loan Status

```
sns.violinplot(x=' loan_status', y=' cibil_score', data=df)
```

```
<Axes: xlabel=' loan_status', ylabel=' cibil_score'>
```

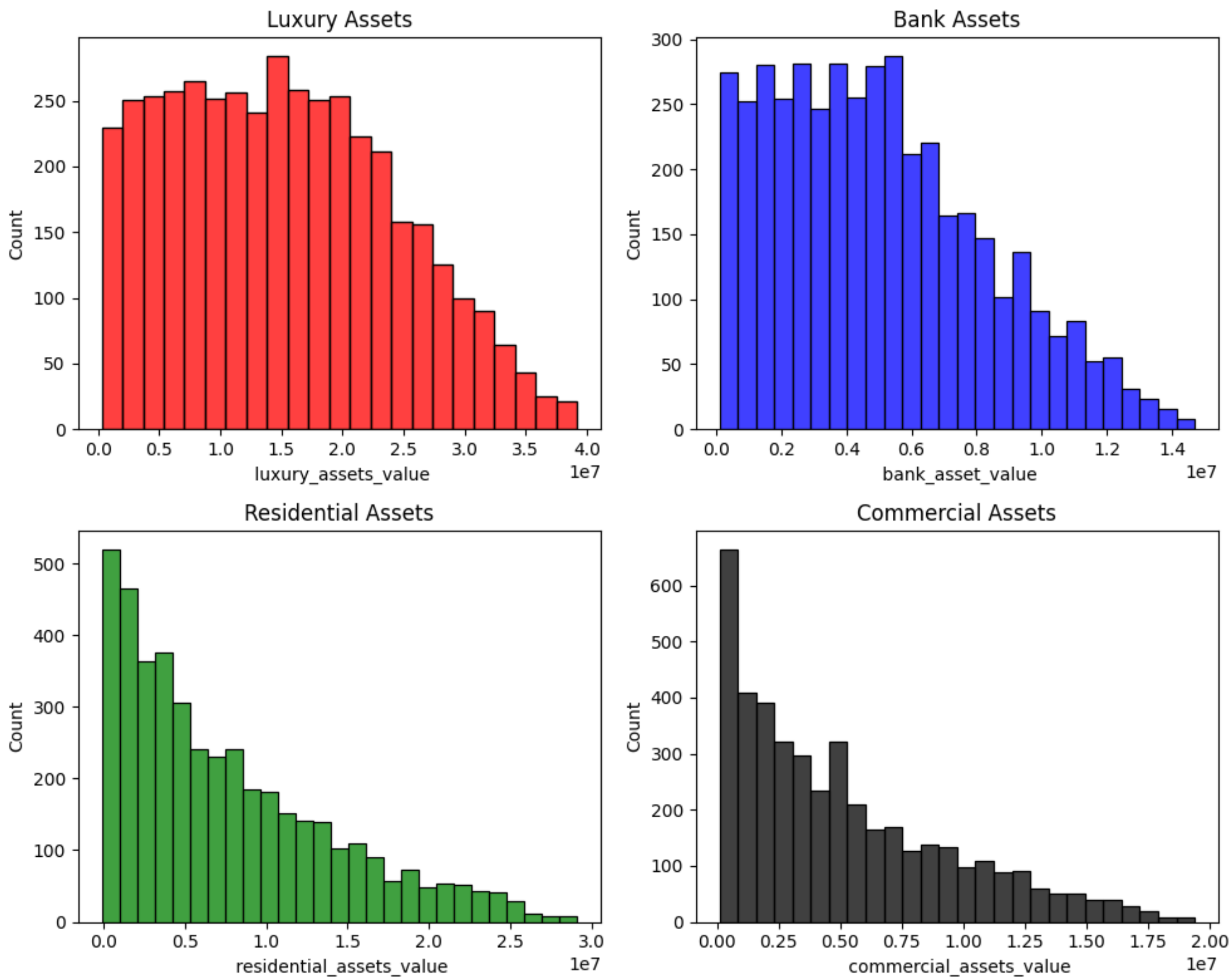


The graph with the shapes (violinplot) clearly shows that people who got their loans approved tend to have higher CIBIL scores, mostly above 600. But for those whose loans weren't approved, the scores are more spread out and usually lower than 550. This means having a higher CIBIL score, especially over 600, really boosts the chances of getting a loan approved. It is very clear that a good CIBIL score is important for loan approval.

Asset Distribution

```
fig, ax = plt.subplots(2, 2, figsize=(10, 8))

plt.subplot(2, 2, 1)
sns.histplot(df[' luxury_assets_value'], color='red')
plt.title("Luxury Assets")

plt.subplot(2, 2, 2)
sns.histplot(df[' bank_asset_value'], color='blue')
plt.title("Bank Assets")

plt.subplot(2, 2, 3)
sns.histplot(df[' residential_assets_value'], color='green')
plt.title("Residential Assets")

plt.subplot(2, 2, 4)
sns.histplot(df[' commercial_assets_value'], color='black')
```

```
plt.title("Commercial Assets")

plt.tight_layout()
plt.show()
```

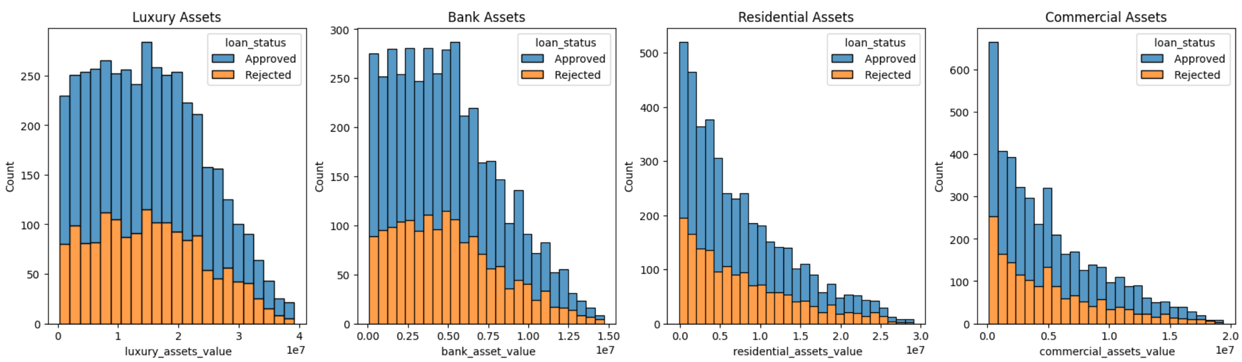


These graphs tell us that most people have lower-valued assets, and the number of people with more valuable assets decreases. It helps us understand how assets affect loan decisions.

Assets vs Loan Status

```
fig, ax = plt.subplots(1, 4, figsize=(20, 5))

sns.histplot(x=' luxury_assets_value', data=df, ax=ax[0], hue='
loan_status', multiple='stack')
ax[0].set_title("Luxury Assets")

sns.histplot(x=' bank_asset_value', data=df, ax=ax[1], hue='
loan_status', multiple='stack')
ax[1].set_title("Bank Assets")
```

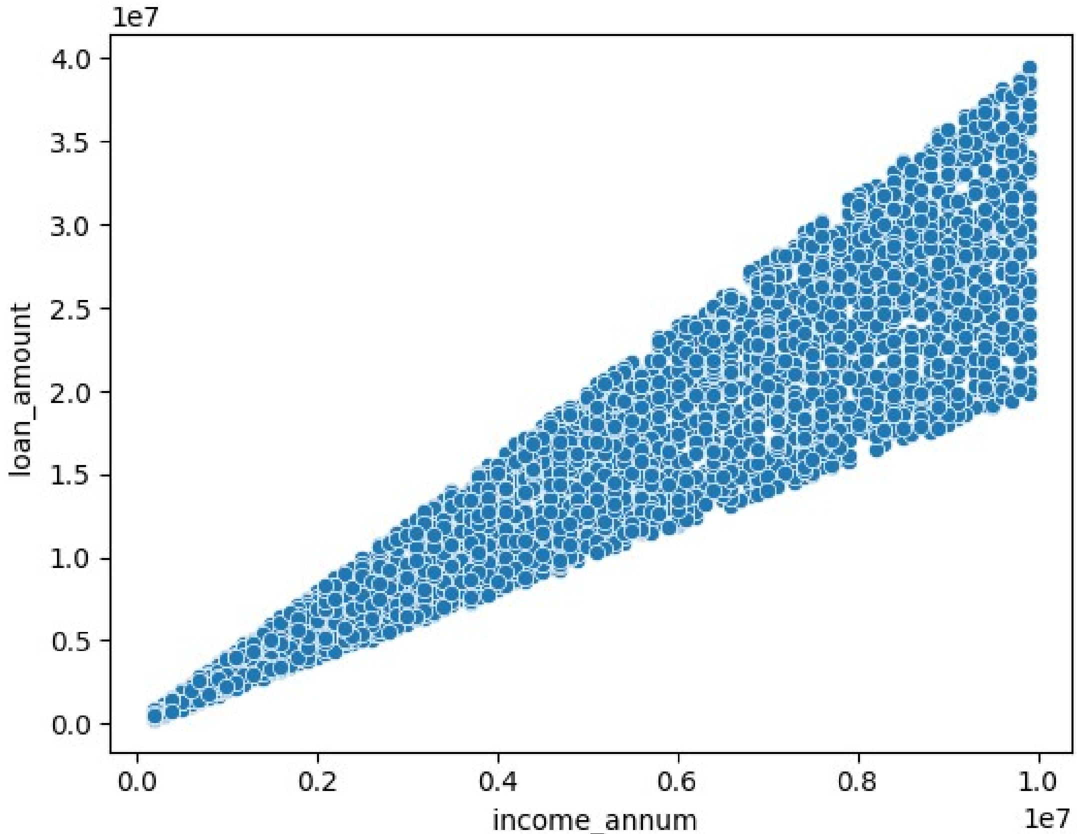
```
sns.histplot(x=' residential_assets_value', data=df, ax=ax[2], hue='
loan_status', multiple='stack')
ax[2].set_title("Residential Assets")

sns.histplot(x=' commercial_assets_value', data=df, ax=ax[3], hue='
loan_status', multiple='stack')
ax[3].set_title("Commercial Assets")

plt.show()
```



Assets offer a safety net for the bank when giving out loans. Both graphs indicate that as assets increase, the likelihood of loan approval slightly goes up, and the chances of rejection decrease

Loan Amount vs Income

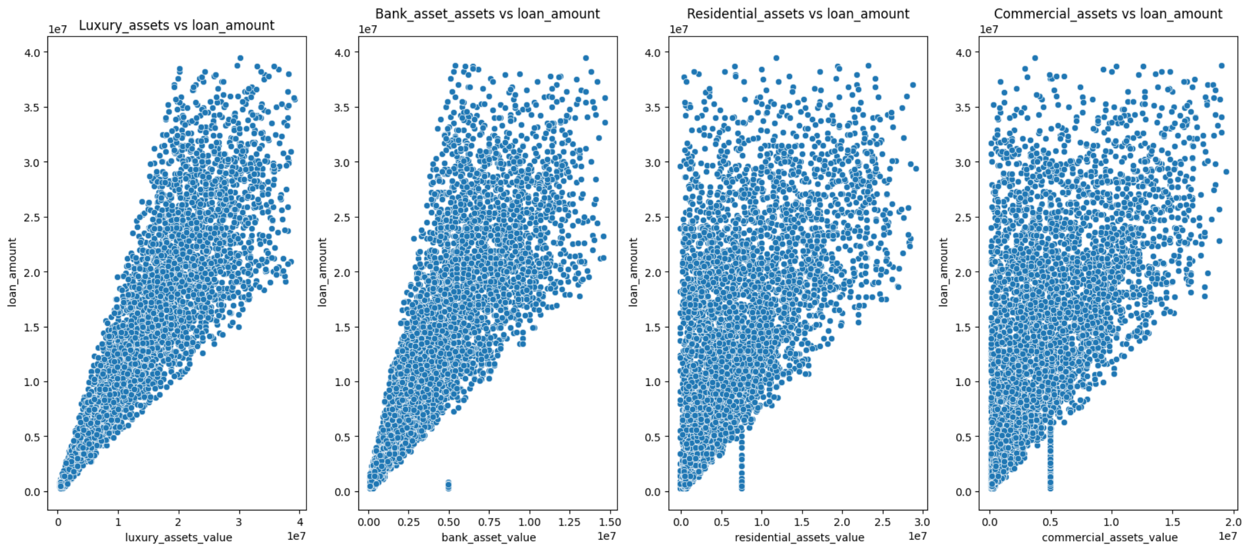
```
sns.scatterplot(x=' income_annum', y = ' loan_amount', data = df)

<Axes: xlabel=' income_annum', ylabel=' loan_amount'>
```



The loan amount and the applicant's annual income share a straightforward connection. When the income is higher, the loan amount tends to be higher as well. This is because the applicant's income plays a major role in determining the appropriate loan amount they can afford to repay.

Assets vs Loan Amount

```
fig, ax = plt.subplots(1,4,figsize=(20, 8))
sns.scatterplot(x=' luxury_assets_value', y = ' loan_amount', data =
df, ax=ax[0]).set_title('Luxury_assets vs loan_amount')
sns.scatterplot(x=' bank_asset_value', y = ' loan_amount', data = df,
ax=ax[1]).set_title('Bank_asset_assets vs loan_amount')
sns.scatterplot(x=' residential_assets_value', y = ' loan_amount',
data = df, ax=ax[2]).set_title('Residential_assets vs loan_amount')
sns.scatterplot(x=' commercial_assets_value', y = ' loan_amount', data
= df, ax=ax[3]).set_title('Commercial_assets vs loan_amount')

Text(0.5, 1.0, 'Commercial_assets vs loan_amount')
```



It is showing that having more assets increases the likelihood of getting a larger loan from the bank. There are some outliers as well

Label Encoding the categorical variables

```
# Label Encoding
df[' education'] = df[' education'].map({' Not Graduate':0, ' 
Graduate':1})
df[' self_employed'] = df[' self_employed'].map({' No':0, ' Yes':1})
df[' loan_status'] = df[' loan_status'].map({' Rejected':0, ' 
Approved':1})
```

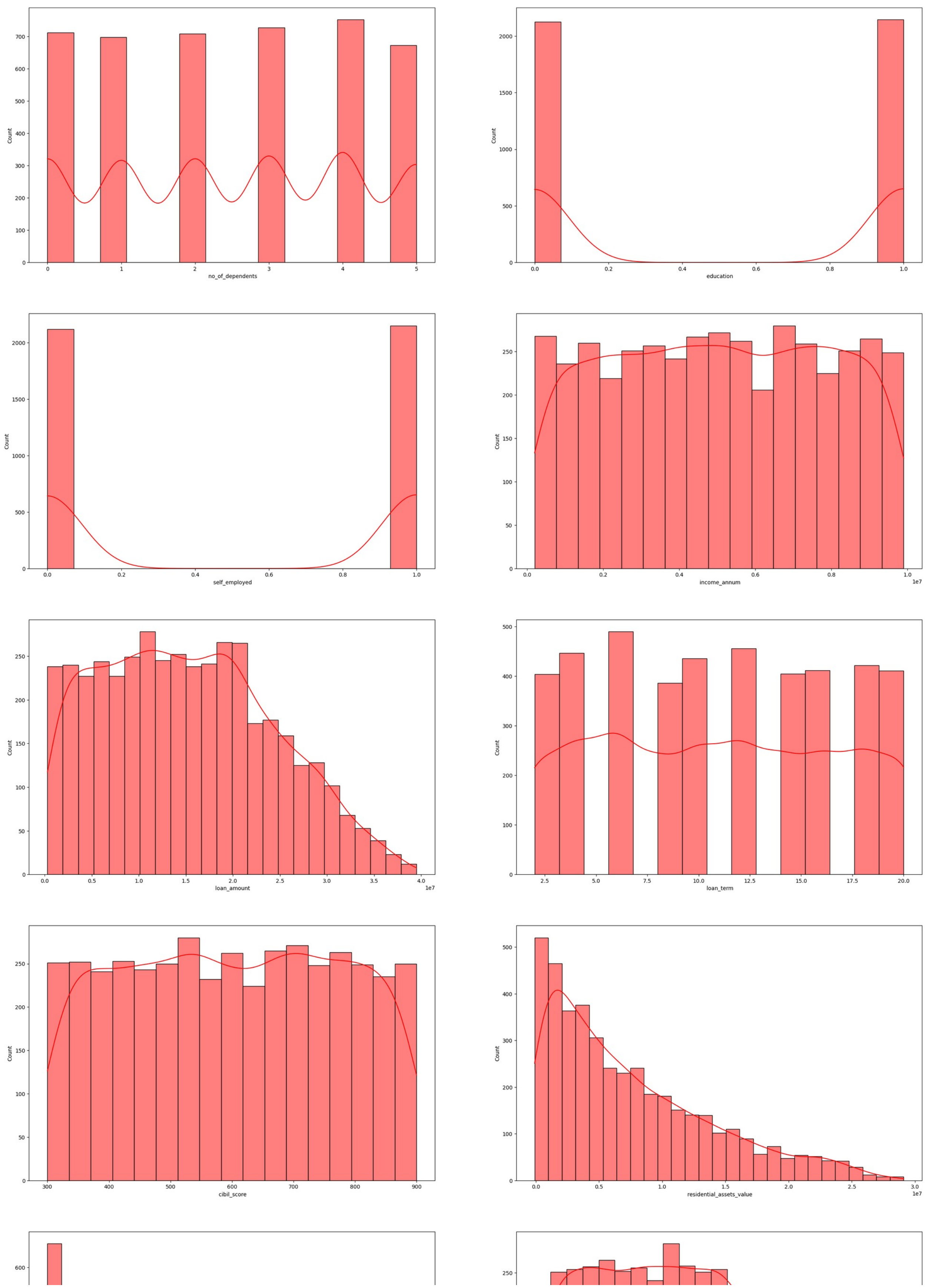
Now all features are numerical

```
df.head()

    no_of_dependents   education   self_employed   income_annum  
loan_amount  \
0                  2           1               0        9600000
29900000
1                  0           0               1        4100000
12200000
2                  3           1               0        9100000
29700000
3                  3           1               0        8200000
30700000
4                  5           0               1        9800000
24200000

    loan_term   cibil_score   residential_assets_value  \
0          12           778                  2400000.0
1           8           417                  2700000.0
2          20           506                  7100000.0
3           8           467                 18200000.0
```

```
4          20           382                  12400000.0

   commercial_assets_value   luxury_assets_value
bank_asset_value  \
0               17600000.0              22700000         8000000.0

1                2200000.0               8800000         3300000.0

2                4500000.0              33300000        12800000.0

3                3300000.0              23300000         7900000.0

4                8200000.0              29400000         5000000.0


   loan_status
0            1
1            0
2            0
3            0
4            0
```

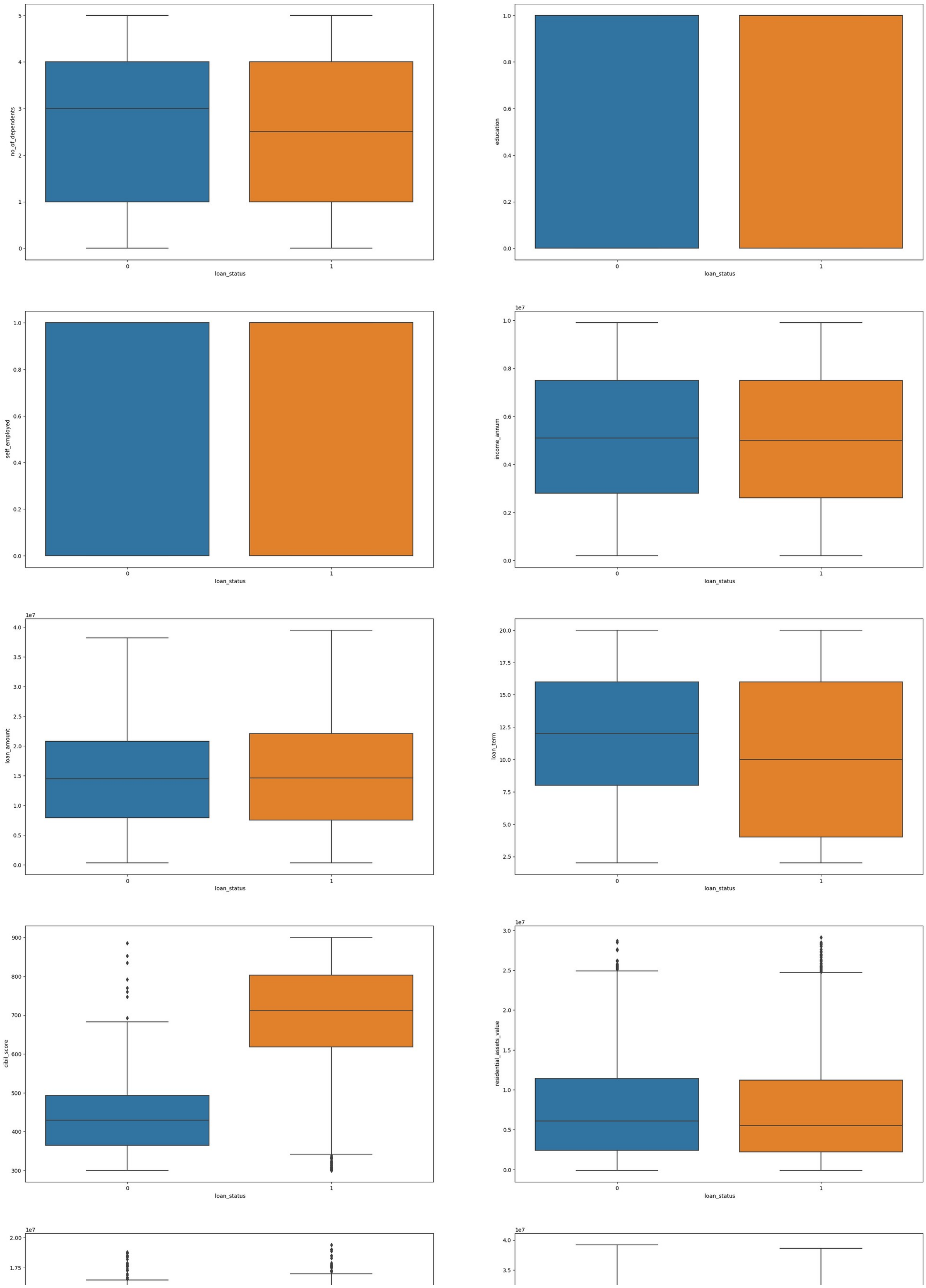
Histograms for each feature

```
fig, axes = plt.subplots(nrows = 5, ncols = 2)
axes = axes.flatten()
fig.set_size_inches(30,50)

for ax, col in zip(axes, df.columns):
    sns.histplot(df[col],kde=True, color='red', ax = ax)
```


Count
no_of_dependents
education
self_employed
income_annum
1e7
loan_amount
loan_term
cibil_score
residential_assets_value

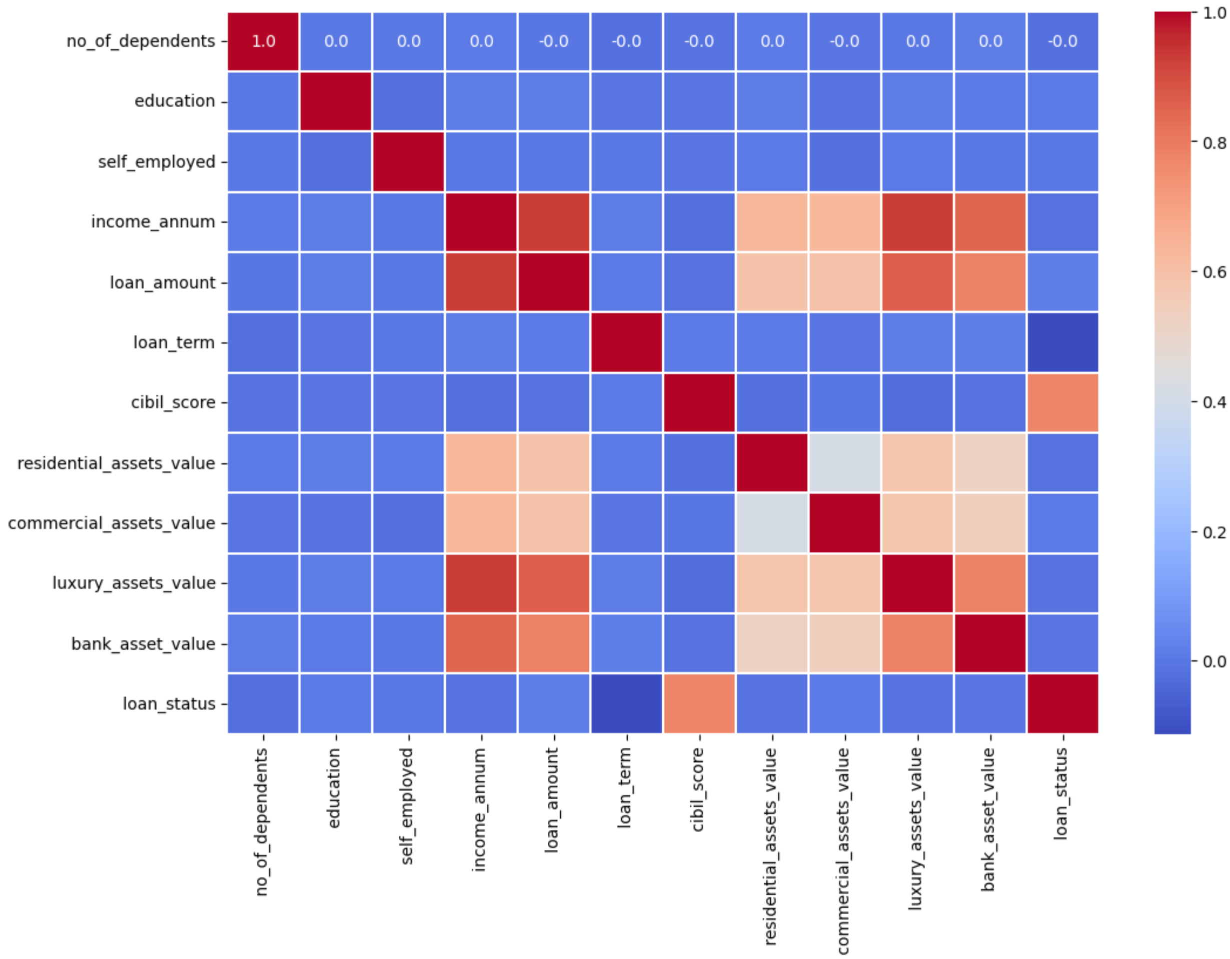
Boxplot for each feature

```
fig, axes = plt.subplots(nrows = 5, ncols = 2)
axes = axes.flatten()
fig.set_size_inches(30,50)

for ax, col in zip(axes, df.columns):
  sns.boxplot(x=' loan_status',y=df[col], ax = ax , data=df)
```


no_of_dependents
loan_status
0
1
education
loan_status
self_employed
loan_status
income_annum
1e7
loan_status
loan_amount
1e7
loan_status
loan_term
loan_status
cibil_score
loan_status
residential_assets_value
1e7
loan_status
1e7
1e7

Correlation Matrix

```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), cmap='coolwarm', annot=True, fmt='.1f',
linewidths=.1)
plt.show()
```



The heatmap of correlation values shows several strong connections:

1. **Movable Assets and Immovable Assets**
2. **Income and Movable Assets**
3. **Income and Immovable Assets**
4. **Movable Assets and Loan Amount**
5. **Immovable Assets and Loan Amount**
6. **Loan Status and Cibil Score**
7. **Loan Amount and Income**

It makes sense that movable and immovable assets are related since they're both types of assets. Similarly, income is linked to both movable and immovable assets, as those with higher income tend to have more assets.

Now, let's look at how assets relate to the loan amount, as well as how income connects to the loan amount. We've already discussed the connection between loan status and CIBIL score in the previous part.

```
df.corr()[' loan_status']

 no_of_dependents           -0.018114
 education                   0.004918
 self_employed               0.000345
 income_annum               -0.015189
 loan_amount                 0.016150
 loan_term                  -0.113036
 cibil_score                 0.770518
 residential_assets_value   -0.014467
 commercial_assets_value     0.007511
 luxury_assets_value        -0.015465
 bank_asset_value           -0.006777
 loan_status                 1.000000
Name:  loan_status, dtype: float64
```

5. Data Preprocessing

```
df

       no_of_dependents   education   self_employed   income_annum  \
0                     2           1               0        9600000
1                     0           0               1        4100000
2                     3           1               0        9100000
3                     3           1               0        8200000
4                     5           0               1        9800000
...                 ...         ...             ...            ...
4264                  5           1               1        1000000
4265                  0           0               1        3300000
4266                  2           0               0        6500000
4267                  1           0               0        4100000
4268                  1           1               0        9200000

       loan_amount   loan_term   cibil_score
residential_assets_value  \
0         29900000          12           778
2400000.0
1         12200000           8           417
2700000.0
2         29700000          20           506
7100000.0
3         30700000           8           467
18200000.0
4         24200000          20           382
```

```
12400000.0
...            ...         ...           ...                       ..
.
4264       2300000          12           317
2800000.0
4265      11300000          20           559
4200000.0
4266      23900000          18           457
1200000.0
4267      12800000           8           780
8200000.0
4268      29700000          10           607
17800000.0

       commercial_assets_value   luxury_assets_value
bank_asset_value  \
0                   17600000.0              22700000
8000000.0
1                    2200000.0               8800000
3300000.0
2                    4500000.0              33300000
12800000.0
3                    3300000.0              23300000
7900000.0
4                    8200000.0              29400000
5000000.0
...                        ...                   ...                  ..
.
4264                  500000.0               3300000
800000.0
4265                 2900000.0              11000000
1900000.0
4266                12400000.0              18100000
7300000.0
4267                  700000.0              14100000
5800000.0
4268                11800000.0              35700000
12000000.0

       loan_status
0                1
1                0
2                0
3                0
4                0
...            ...
4264             0
4265             1
4266             0
```

```
4267            1
4268            1

[4269 rows x 12 columns]
```

Outlier Detection

Using zscore

```
from scipy.stats import zscore

# Create a copy of the DataFrame to avoid modifying the original
df_copy = df.copy()

# Calculate Z-scores for each numeric column
numeric_columns = df_copy.select_dtypes(include=[np.number]).columns
df_copy[numeric_columns] = df_copy[numeric_columns].apply(zscore)

# Set a threshold for Z-score (e.g., 3)
threshold = 3

# Identify outliers based on Z-score
outliers = df_copy[(np.abs(df_copy[numeric_columns]) > 
threshold).any(axis=1)]

print(outliers.count())
```

```
 no_of_dependents            33
 education                   33
 self_employed               33
 income_annum                33
 loan_amount                 33
 loan_term                   33
 cibil_score                 33
 residential_assets_value    33
 commercial_assets_value     33
 luxury_assets_value         33
 bank_asset_value            33
 loan_status                 33
dtype: int64
```

Using IsolationForest

```
from sklearn.ensemble import IsolationForest

# Create an Isolation Forest model
clf = IsolationForest(contamination='auto', random_state=42)  # Adjust 
contamination based on your data

# Fit the model and predict outliers
```

```
df_copy['outlier'] = clf.fit_predict(df_copy)

# Count the number of outlier rows
outlier_count = df_copy[df_copy['outlier'] == -1].shape[0]

# Display the count of outliers
print("Number of outlier rows:", outlier_count)
```

```
Number of outlier rows: 2706
```

```
X = df.drop(' loan_status', axis = 1)
y = df[' loan_status']

y.value_counts()
```

```
1    2656
0    1613
Name:  loan_status, dtype: int64
```

It is clearly unbalanced data, so we need to oversample the minority class

Oversampling the minority class

```
from imblearn.over_sampling import RandomOverSampler

rs = RandomOverSampler()

X, y = rs.fit_resample(X,y)

y.value_counts()
```

```
1    2656
0    2656
Name:  loan_status, dtype: int64
```

```
import warnings

# Suppress warnings within this code block
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
```

6. ML Modelling with Hyperparameter Tuning

```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.compose import ColumnTransformer
from sklearn import tree
```

```python
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
```

Train Test Split

```python
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Feature Scaling

```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Save the fitted scaler to a file using pickle
scaler_filename = 'standard_scaler.pkl'
with open(scaler_filename, 'wb') as scaler_file:
    pickle.dump(sc, scaler_file)
```

1. Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier

# Define the model
rf_model = RandomForestClassifier()

# Define hyperparameters for tuning
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_depth': [None, 10, 15, 20],
    'min_samples_split': [2, 5, 10]
}

# Perform GridSearchCV
grid_search_rf = GridSearchCV(rf_model, param_grid, cv=5)
grid_search_rf.fit(X_train, y_train)

# Print the best hyperparameters
best_hyperparameters = grid_search_rf.best_params_
print("Best Hyperparameters (Random Forest):", best_hyperparameters)

# Evaluate the model
train_accuracy = accuracy_score(y_train,
grid_search_rf.predict(X_train))
test_accuracy = accuracy_score(y_test, grid_search_rf.predict(X_test))

print("Random Forest Classifier:")
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

```
Best Hyperparameters (Random Forest): {'max_depth': 15,
'min_samples_split': 2, 'n_estimators': 100}
Random Forest Classifier:
Training Accuracy: 1.0
Testing Accuracy: 0.9811853245531514
```

2. Support Vector Classification (SVC)

```
from sklearn.svm import SVC

# Define the model
svm_model = SVC()

# Define hyperparameters for tuning
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
}

# Perform GridSearchCV
grid_search_svm = GridSearchCV(svm_model, param_grid, cv=5)
grid_search_svm.fit(X_train, y_train)

# Print the best hyperparameters
best_hyperparameters = grid_search_svm.best_params_
print("Best Hyperparameters (SVM):", best_hyperparameters)

# Evaluate the model
train_accuracy = accuracy_score(y_train,
grid_search_svm.predict(X_train))
test_accuracy = accuracy_score(y_test,
grid_search_svm.predict(X_test))

print("SVM Classifier:")
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

```
Best Hyperparameters (SVM): {'C': 10, 'kernel': 'rbf'}
SVM Classifier:
Training Accuracy: 0.9809366909861144
Testing Accuracy: 0.9567262464722484
```

3. Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

# Define the model
nb_model = GaussianNB()

# No hyperparameters to tune for Gaussian Naive Bayes
```

```
# Fit the model
nb_model.fit(X_train, y_train)

# Evaluate the model
train_accuracy = accuracy_score(y_train, nb_model.predict(X_train))
test_accuracy = accuracy_score(y_test, nb_model.predict(X_test))

print("Naive Bayes Classifier:")
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

```
Naive Bayes Classifier:
Training Accuracy: 0.9512826547422923
Testing Accuracy: 0.9529633113828786
```

4. Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier

# Define the model
gb_model = GradientBoostingClassifier()

# Define hyperparameters for tuning
param_grid = {
    'n_estimators': [10, 50, 100],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}

# Perform GridSearchCV
grid_search_gb = GridSearchCV(gb_model, param_grid, cv=5)
grid_search_gb.fit(X_train, y_train)

# Print the best hyperparameters
best_hyperparameters = grid_search_gb.best_params_
print("Best Hyperparameters (GradientBoostingClassifier):",
best_hyperparameters)

# Evaluate the model
train_accuracy = accuracy_score(y_train,
grid_search_gb.predict(X_train))
test_accuracy = accuracy_score(y_test, grid_search_gb.predict(X_test))

print("Gradient Boosting Classifier:")
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

```
Best Hyperparameters (GradientBoostingClassifier): {'learning_rate':
0.1, 'max_depth': 5, 'n_estimators': 100}
```

```
Gradient Boosting Classifier:
Training Accuracy: 1.0
Testing Accuracy: 0.9915333960489181
```

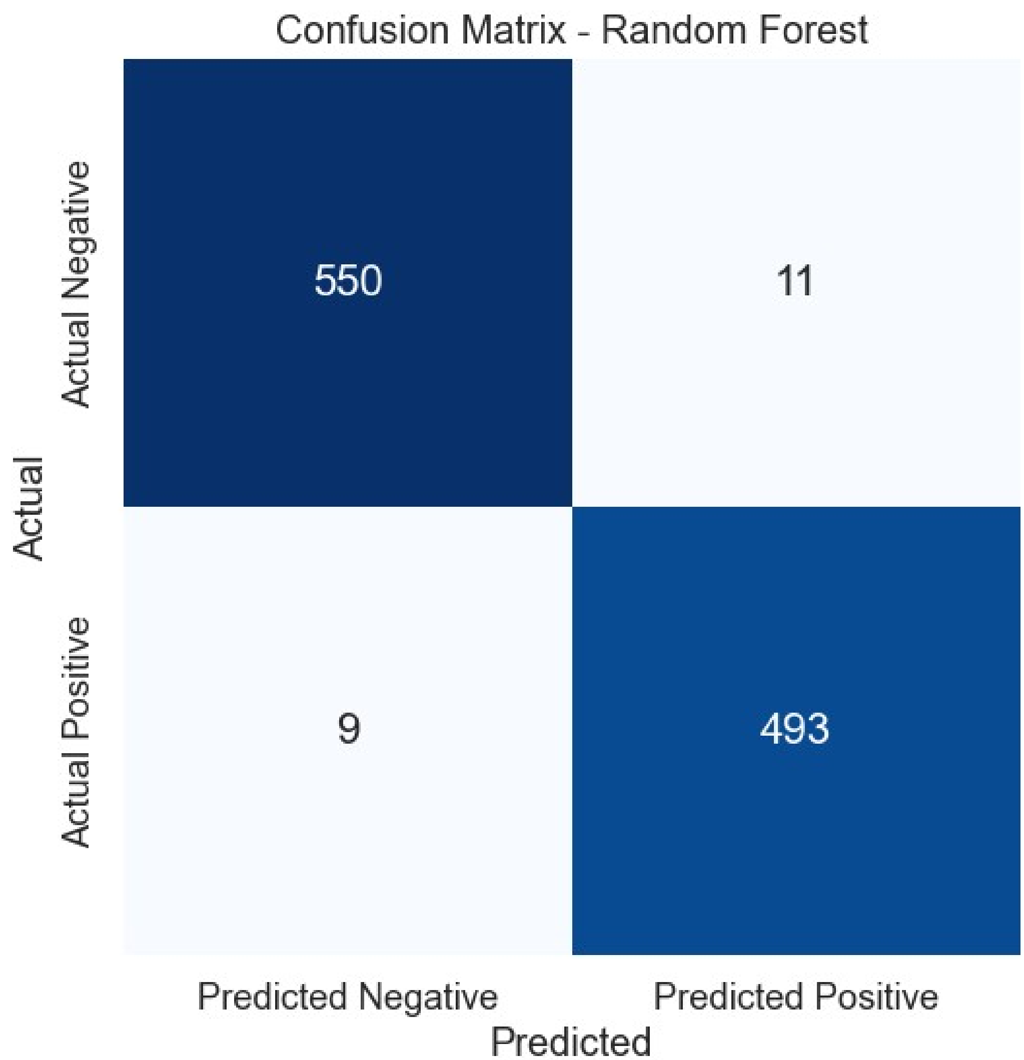
5. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import GridSearchCV

# Define the model
dt_model = DecisionTreeClassifier()

# Define hyperparameters for tuning
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform GridSearchCV
grid_search_dc = GridSearchCV(dt_model, param_grid, cv=5)
grid_search_dc.fit(X_train, y_train)

# Print the best hyperparameters
best_hyperparameters = grid_search_dc.best_params_
print("Best Hyperparameters (Decision Tree):", best_hyperparameters)

# Get the best model
best_dt_model = grid_search_dc.best_estimator_

# Evaluate the model
train_accuracy = accuracy_score(y_train,
best_dt_model.predict(X_train))
test_accuracy = accuracy_score(y_test, best_dt_model.predict(X_test))

print("Decision Tree Classifier:")
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

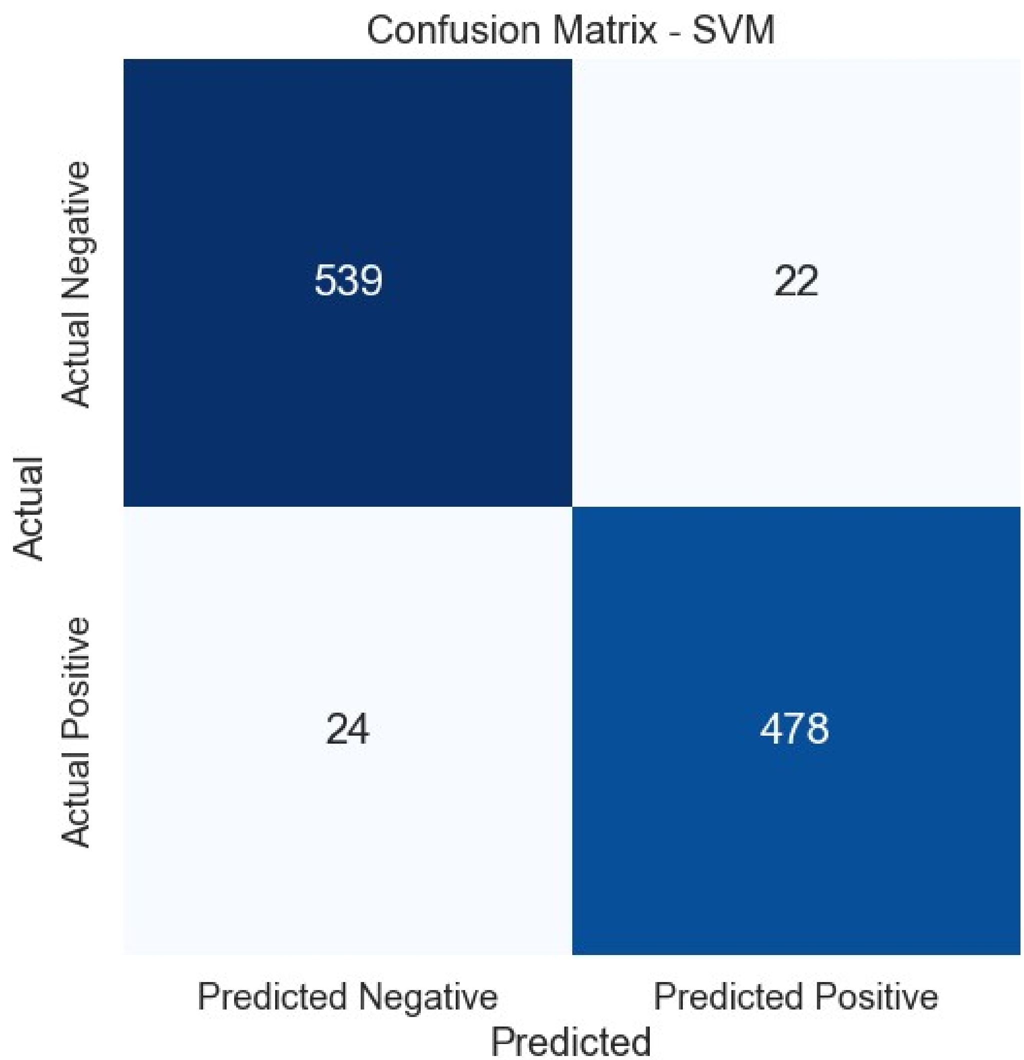
```
Best Hyperparameters (Decision Tree): {'criterion': 'gini',
'max_depth': 30, 'min_samples_leaf': 4, 'min_samples_split': 2}
Decision Tree Classifier:
Training Accuracy: 0.9922334666980466
Testing Accuracy: 0.9774223894637818
```

8. Model Evalution (Confusion Matrix and Classification Report)

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of models
models = [
    ("Random Forest", grid_search_rf),
    ("SVM", grid_search_svm),
    ("Naive Bayes", nb_model),
    ("Gradient Boosting", grid_search_gb),
    ("Decision Tree", grid_search_dc),
]

# Loop through each model
for model_name, model in models:
    # Get model predictions
    predictions = model.predict(X_test)

    # Calculate confusion matrix
    cm = confusion_matrix(y_test, predictions)

    # Create a confusion matrix heatmap
    plt.figure(figsize=(8, 6))
    sns.set(font_scale=1.2)  # Adjust font size
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                annot_kws={"size": 16}, square=True,
                xticklabels=['Predicted Negative', 'Predicted
Positive'],
                yticklabels=['Actual Negative', 'Actual Positive'])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()

    # Display classification report
    print(f"Classification Report - {model_name}:\n")
    print(classification_report(y_test, predictions))
```

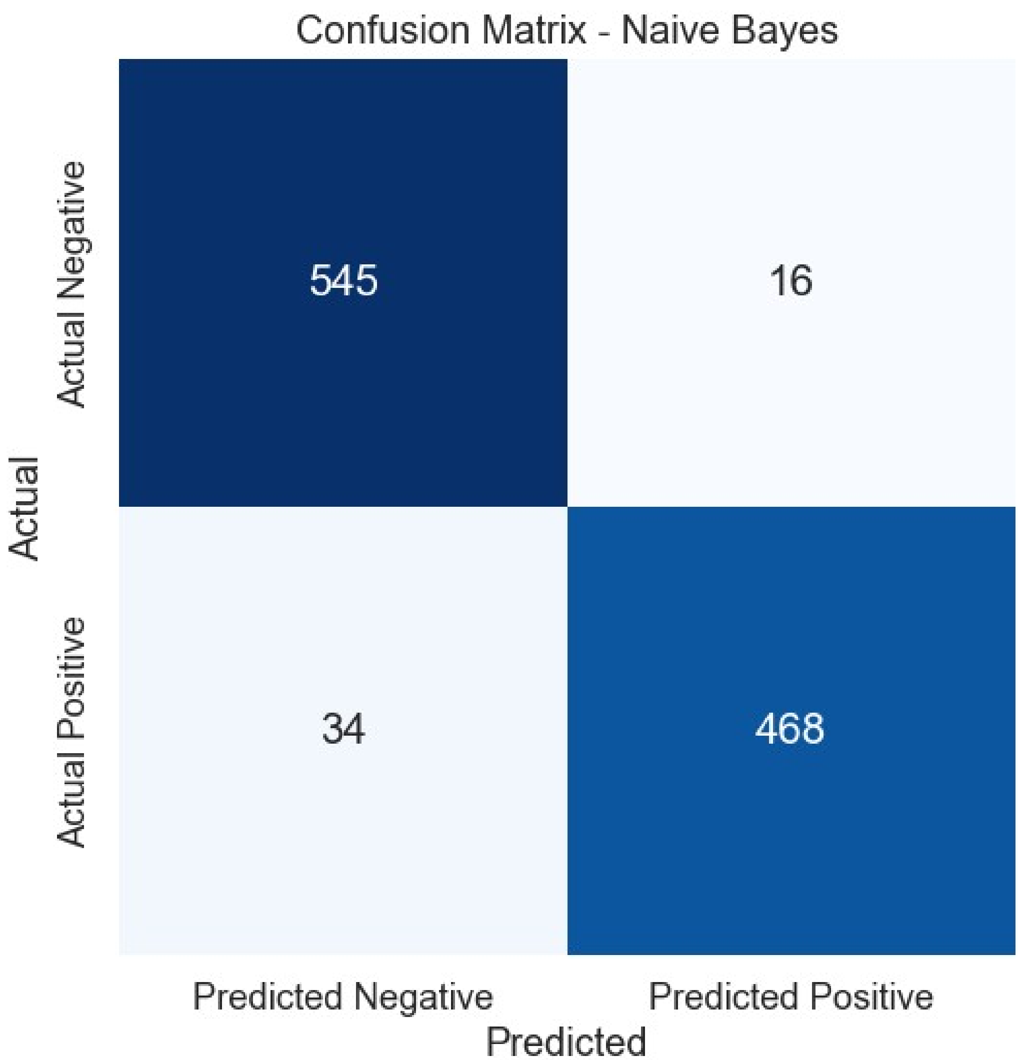



```
Classification Report - Random Forest:

              precision    recall  f1-score   support

           0       0.98      0.98      0.98       561
           1       0.98      0.98      0.98       502

    accuracy                           0.98      1063
   macro avg       0.98      0.98      0.98      1063
weighted avg       0.98      0.98      0.98      1063
```

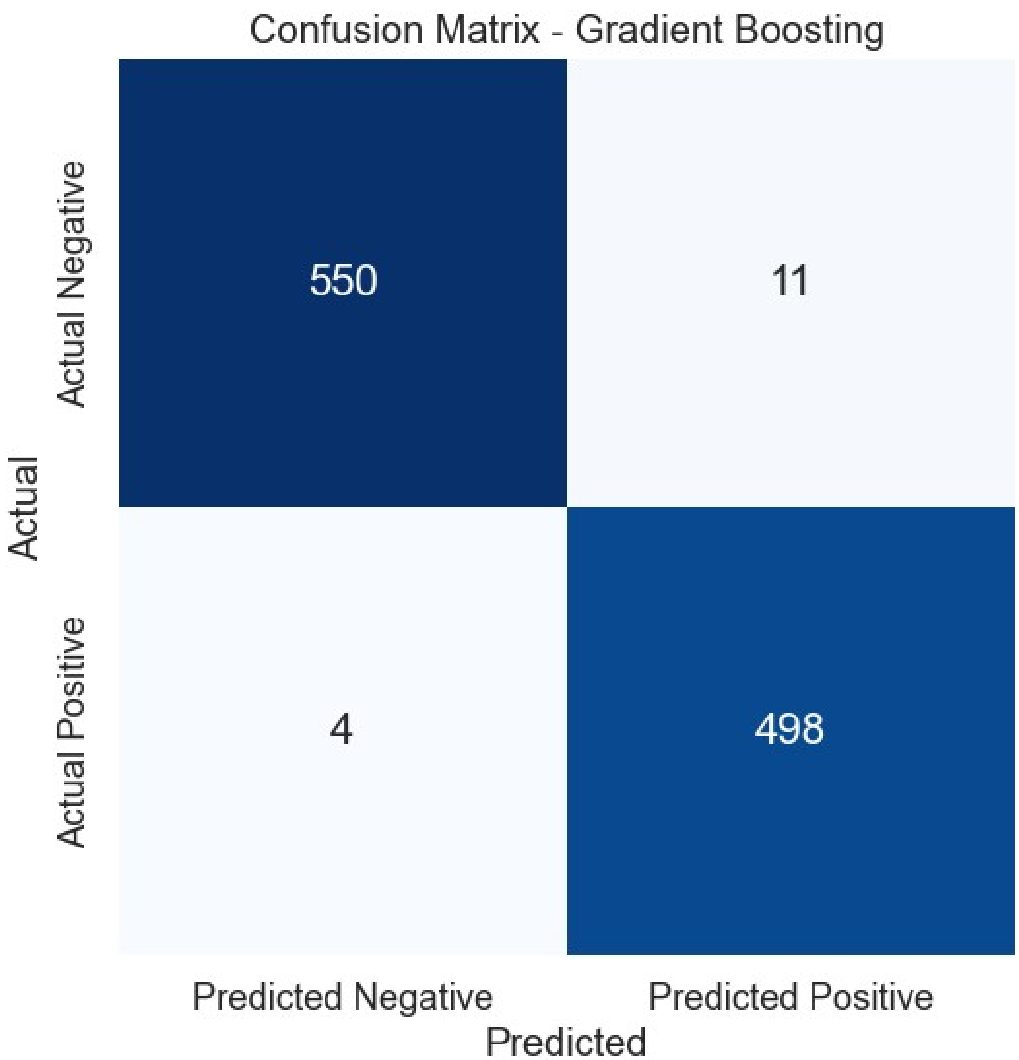


Classification Report - SVM:

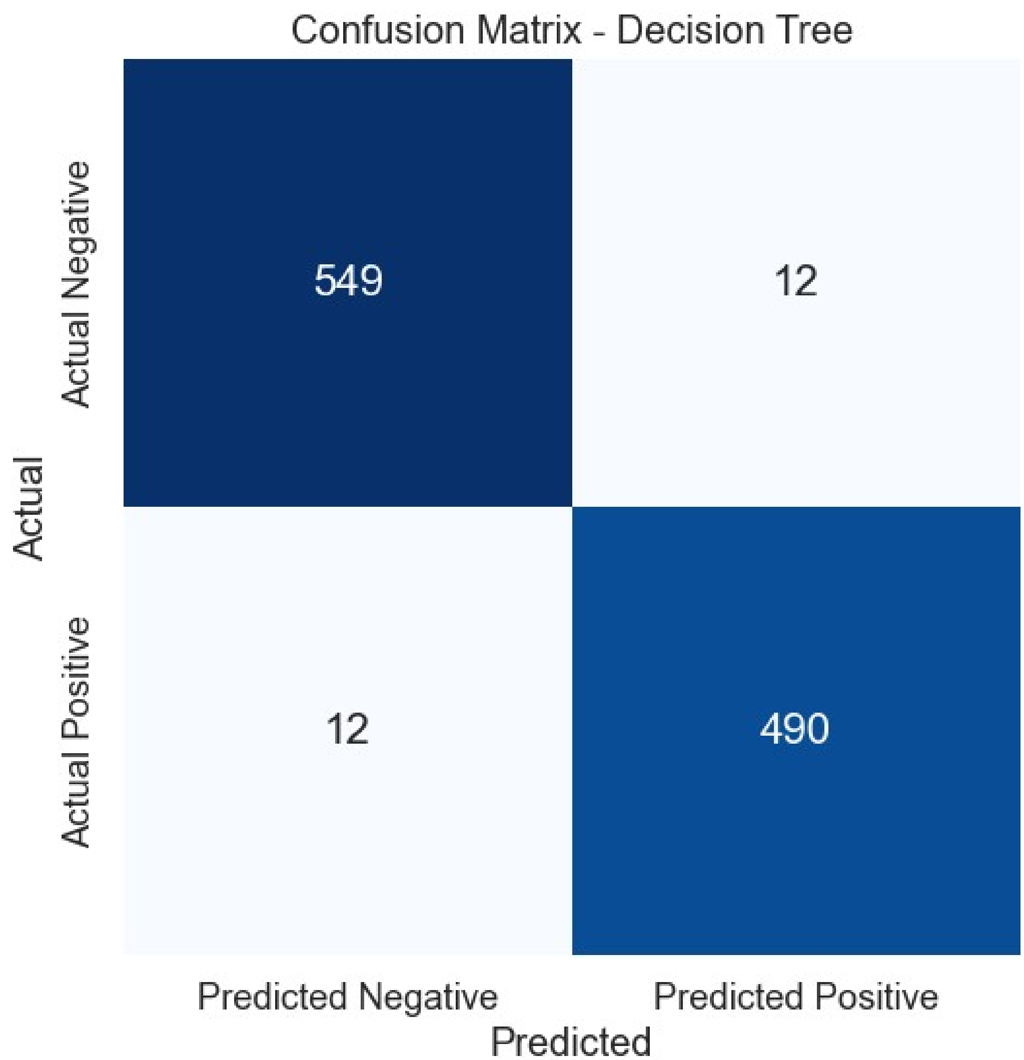
	precision	recall	f1-score	support
0	0.96	0.96	0.96	561
1	0.96	0.95	0.95	502
accuracy			0.96	1063
macro avg	0.96	0.96	0.96	1063
weighted avg	0.96	0.96	0.96	1063




```
Classification Report - Naive Bayes:

              precision    recall  f1-score   support

           0       0.94      0.97      0.96       561
           1       0.97      0.93      0.95       502

    accuracy                           0.95      1063
   macro avg       0.95      0.95      0.95      1063
weighted avg       0.95      0.95      0.95      1063
```




```
Classification Report - Gradient Boosting:

              precision    recall  f1-score   support

           0       0.99      0.98      0.99       561
           1       0.98      0.99      0.99       502

    accuracy                           0.99      1063
   macro avg       0.99      0.99      0.99      1063
weighted avg       0.99      0.99      0.99      1063
```




```
Classification Report - Decision Tree:

              precision    recall  f1-score   support

           0       0.98      0.98      0.98       561
           1       0.98      0.98      0.98       502

    accuracy                           0.98      1063
   macro avg       0.98      0.98      0.98      1063
weighted avg       0.98      0.98      0.98      1063
```

Summary of Model Performance for Loan Approval Prediction

When looking at different ways to predict if loans will be approved or not, we found that the Random Forest and Gradient Boosting performedworked really well. It was accurate and could predict outcomes quite accurately. Decision tree model also did a good job.

However, Support Vector Machine (SVM) and Naive Bayes didn't work well like the above models. They didn't predict as accurately as the Decision Tree and Random Forest models.

Saving the Gradient Boosting model

```
import pickle

# Save the grid_search_rf model to a file using pickle
model_filename = 'grid_search_gb_model.pkl'
with open(model_filename, 'wb') as model_file:
    pickle.dump(grid_search_gb, model_file)
```

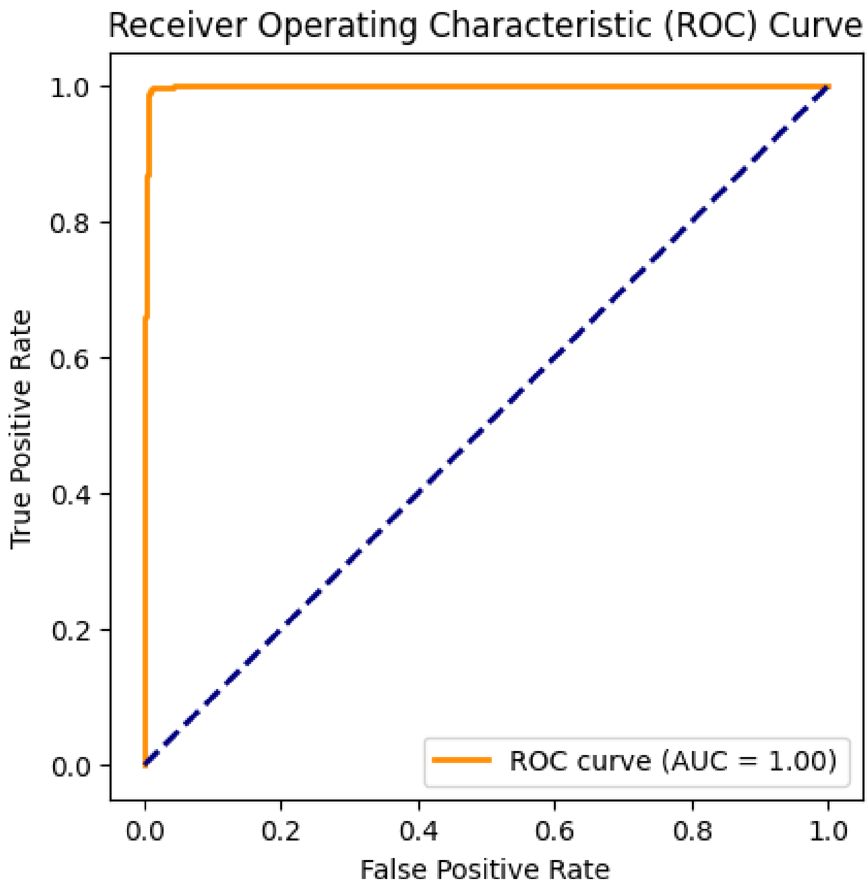
Loading the saved model

```
# Load the saved model from the file
model_filename = 'grid_search_gb_model.pkl'
with open(model_filename, 'rb') as model_file:
    loaded_model = pickle.load(model_file)

loaded_model
```

```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'learning_rate': [0.01, 0.1, 0.2],
                         'max_depth': [3, 4, 5],
                         'n_estimators': [10, 50, 100]})
```

ROC curve for Gradient Boosting model

```
from sklearn.metrics import roc_curve, auc

# Get predicted probabilities for the positive class
y_probabilities = loaded_model.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_probabilities)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(5, 5))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
ROC curve (AUC = 1.00)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0