# Library Importation

In [1]:
```
# Importing necessary libraries
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import os
import sys
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, cross_va
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB, ComplementNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.metrics import precision_score
from sklearn import metrics
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve, roc_auc_score
from imblearn.over_sampling import SMOTE as imblearn_SMOTE
import matplotlib.pyplot as plt
import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings("ignore", category=ConvergenceWarning)
from imblearn.over_sampling import SMOTE
```

# Data Importation

In [2]:
```
#Dataset Importation
df = pd.read_csv("Bank_Personal_Loan.csv")
```

# Initial Data Analysis

In [3]:
```
# Checking head of dataset (first 10 rows of the dataset).
df.head(n=10)
```

Out[3]:

| | ID | Age | Experience | Income | ZIP Code | Family | CCAvg | Education | Mortgage | Personal Loan | Securitie Accoun |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 25 | 1 | 49 | 91107 | 4 | 1/60 | 1 | 0 | 0 | |
| **1** | 2 | 45 | 19 | 34 | 90089 | 3 | 1/50 | 1 | 0 | 0 | |
| **2** | 3 | 39 | 15 | 11 | 94720 | 1 | 1/00 | 1 | 0 | 0 | |
| **3** | 4 | 35 | 9 | 100 | 94112 | 1 | 2/70 | 2 | 0 | 0 | |
| **4** | 5 | 35 | 8 | 45 | 91330 | 4 | 1/00 | 2 | 0 | 0 | |
| **5** | 6 | 37 | 13 | 29 | 92121 | 4 | 0/40 | 2 | 155 | 0 | |
| **6** | 7 | 53 | 27 | 72 | 91711 | 2 | 1/50 | 2 | 0 | 0 | |
| **7** | 8 | 50 | 24 | 22 | 93943 | 1 | 0/30 | 3 | 0 | 0 | |
| **8** | 9 | 35 | 10 | 81 | 90089 | 3 | 0/60 | 2 | 104 | 0 | |
| **9** | 10 | 34 | 9 | 180 | 93023 | 1 | 8/90 | 3 | 0 | 1 | |

In [4]:

```
#displaying the shape
df.shape
```

Out[4]: (5000, 14)

In [5]:

```
#basic statistics of the dataset
df.describe().T
```

Out[5]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **ID** | 5000.0 | 2500.5000 | 1443.520003 | 1.0 | 1250.75 | 2500.5 | 3750.25 | 5000.0 |
| **Age** | 5000.0 | 45.3384 | 11.463166 | 23.0 | 35.00 | 45.0 | 55.00 | 67.0 |
| **Experience** | 5000.0 | 20.1046 | 11.467954 | -3.0 | 10.00 | 20.0 | 30.00 | 43.0 |
| **Income** | 5000.0 | 73.7742 | 46.033729 | 8.0 | 39.00 | 64.0 | 98.00 | 224.0 |
| **ZIP Code** | 5000.0 | 93152.5030 | 2121.852197 | 9307.0 | 91911.00 | 93437.0 | 94608.00 | 96651.0 |
| **Family** | 5000.0 | 2.3964 | 1.147663 | 1.0 | 1.00 | 2.0 | 3.00 | 4.0 |
| **Education** | 5000.0 | 1.8810 | 0.839869 | 1.0 | 1.00 | 2.0 | 3.00 | 3.0 |
| **Mortgage** | 5000.0 | 56.4988 | 101.713802 | 0.0 | 0.00 | 0.0 | 101.00 | 635.0 |
| **Personal Loan** | 5000.0 | 0.0960 | 0.294621 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 |
| **Securities Account** | 5000.0 | 0.1044 | 0.305809 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 |
| **CD Account** | 5000.0 | 0.0604 | 0.238250 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 |
| **Online** | 5000.0 | 0.5968 | 0.490589 | 0.0 | 0.00 | 1.0 | 1.00 | 1.0 |
| **CreditCard** | 5000.0 | 0.2940 | 0.455637 | 0.0 | 0.00 | 0.0 | 1.00 | 1.0 |

In [6]:

```
# Checking for duplicates
df.duplicated().sum()
```

Out[6]: 0

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   ID                  5000 non-null   int64 
 1   Age                 5000 non-null   int64 
 2   Experience          5000 non-null   int64 
 3   Income              5000 non-null   int64 
 4   ZIP Code            5000 non-null   int64 
 5   Family              5000 non-null   int64 
 6   CCAvg               5000 non-null   object
 7   Education           5000 non-null   int64 
 8   Mortgage            5000 non-null   int64 
 9   Personal Loan       5000 non-null   int64 
 10  Securities Account  5000 non-null   int64 
 11  CD Account          5000 non-null   int64 
 12  Online              5000 non-null   int64 
 13  CreditCard          5000 non-null   int64 
dtypes: int64(13), object(1)
memory usage: 547.0+ KB
```

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

In [9]:
```
# finding unique values in the column
for column_name in df.columns:
    print("********************************************************************", cc
    print()
    print(set(df[column_name].tolist()))
    print()
```

```
******************************************************************* ID ***********
*************************************************************

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2
3, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 6
4, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 10
4, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120,
121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 13
7, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153,
154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 17
0, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186,
187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 20
3, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219,
220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 23
6, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252,
253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 26
9, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,
286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 30
2, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318,
319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 33
5, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351,
352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 36
8, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384,
385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 40
1, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417,
418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 43
4, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450,
451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 46
7, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483,
484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 50
0, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516,
517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 53
3, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549,
550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 56
6, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582,
583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 59
9, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615,
616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 63
2, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648,
649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 66
5, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681,
682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 69
8, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714,
715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 73
1, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747,
748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 76
4, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780,
781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 79
7, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813,
814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 83
0, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846,
847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 86
3, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879,
880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 89
6, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912,
913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 92
9, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945,
946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 96
2, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978,
979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 99
5, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009,
1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 102
```

3, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1
037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050,
1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 106
4, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1
078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091,
1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 110
5, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1
119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1132,
1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 114
6, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1
160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173,
1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 118
7, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1
201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214,
1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1223, 1224, 1225, 1226, 1227, 122
8, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241, 1
242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255,
1256, 1257, 1258, 1259, 1260, 1261, 1262, 1263, 1264, 1265, 1266, 1267, 1268, 126
9, 1270, 1271, 1272, 1273, 1274, 1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1
283, 1284, 1285, 1286, 1287, 1288, 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296,
1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 131
0, 1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1
324, 1325, 1326, 1327, 1328, 1329, 1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337,
1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 135
1, 1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1
365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1378,
1379, 1380, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 139
2, 1393, 1394, 1395, 1396, 1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1
406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419,
1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 143
3, 1434, 1435, 1436, 1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445, 1446, 1
447, 1448, 1449, 1450, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460,
1461, 1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 147
4, 1475, 1476, 1477, 1478, 1479, 1480, 1481, 1482, 1483, 1484, 1485, 1486, 1487, 1
488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 1496, 1497, 1498, 1499, 1500, 1501,
1502, 1503, 1504, 1505, 1506, 1507, 1508, 1509, 1510, 1511, 1512, 1513, 1514, 151
5, 1516, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1
529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542,
1543, 1544, 1545, 1546, 1547, 1548, 1549, 1550, 1551, 1552, 1553, 1554, 1555, 155
6, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 1
570, 1571, 1572, 1573, 1574, 1575, 1576, 1577, 1578, 1579, 1580, 1581, 1582, 1583,
1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 159
7, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1
611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1619, 1620, 1621, 1622, 1623, 1624,
1625, 1626, 1627, 1628, 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636, 1637, 163
8, 1639, 1640, 1641, 1642, 1643, 1644, 1645, 1646, 1647, 1648, 1649, 1650, 1651, 1
652, 1653, 1654, 1655, 1656, 1657, 1658, 1659, 1660, 1661, 1662, 1663, 1664, 1665,
1666, 1667, 1668, 1669, 1670, 1671, 1672, 1673, 1674, 1675, 1676, 1677, 1678, 167
9, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1690, 1691, 1692, 1
693, 1694, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706,
1707, 1708, 1709, 1710, 1711, 1712, 1713, 1714, 1715, 1716, 1717, 1718, 1719, 172
0, 1721, 1722, 1723, 1724, 1725, 1726, 1727, 1728, 1729, 1730, 1731, 1732, 1733, 1
734, 1735, 1736, 1737, 1738, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747,
1748, 1749, 1750, 1751, 1752, 1753, 1754, 1755, 1756, 1757, 1758, 1759, 1760, 176
1, 1762, 1763, 1764, 1765, 1766, 1767, 1768, 1769, 1770, 1771, 1772, 1773, 1774, 1
775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 1783, 1784, 1785, 1786, 1787, 1788,
1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1798, 1799, 1800, 1801, 180
2, 1803, 1804, 1805, 1806, 1807, 1808, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1
816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829,
1830, 1831, 1832, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1840, 1841, 1842, 184
3, 1844, 1845, 1846, 1847, 1848, 1849, 1850, 1851, 1852, 1853, 1854, 1855, 1856, 1
857, 1858, 1859, 1860, 1861, 1862, 1863, 1864, 1865, 1866, 1867, 1868, 1869, 1870,
1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 188
4, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1

```
898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911,
1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 192
5, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1
939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952,
1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 196
6, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1
980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993,
1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 200
7, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2
021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034,
2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 204
8, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2
062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075,
2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 208
9, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2
103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116,
2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 213
0, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2
144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157,
2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 217
1, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2
185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198,
2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 221
2, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2
226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239,
2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 225
3, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2
267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280,
2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 229
4, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2
308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321,
2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 233
5, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2
349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362,
2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 237
6, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2
390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403,
2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 241
7, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2
431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444,
2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 245
8, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2
472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485,
2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 249
9, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2
513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526,
2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 254
0, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2
554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567,
2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 258
1, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2
595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608,
2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 262
2, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2
636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649,
2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 266
3, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2
677, 2678, 2679, 2680, 2681, 2682, 2683, 2684, 2685, 2686, 2687, 2688, 2689, 2690,
2691, 2692, 2693, 2694, 2695, 2696, 2697, 2698, 2699, 2700, 2701, 2702, 2703, 270
4, 2705, 2706, 2707, 2708, 2709, 2710, 2711, 2712, 2713, 2714, 2715, 2716, 2717, 2
718, 2719, 2720, 2721, 2722, 2723, 2724, 2725, 2726, 2727, 2728, 2729, 2730, 2731,
2732, 2733, 2734, 2735, 2736, 2737, 2738, 2739, 2740, 2741, 2742, 2743, 2744, 274
5, 2746, 2747, 2748, 2749, 2750, 2751, 2752, 2753, 2754, 2755, 2756, 2757, 2758, 2
759, 2760, 2761, 2762, 2763, 2764, 2765, 2766, 2767, 2768, 2769, 2770, 2771, 2772,
```

```
2773, 2774, 2775, 2776, 2777, 2778, 2779, 2780, 2781, 2782, 2783, 2784, 2785, 278
6, 2787, 2788, 2789, 2790, 2791, 2792, 2793, 2794, 2795, 2796, 2797, 2798, 2799, 2
800, 2801, 2802, 2803, 2804, 2805, 2806, 2807, 2808, 2809, 2810, 2811, 2812, 2813,
2814, 2815, 2816, 2817, 2818, 2819, 2820, 2821, 2822, 2823, 2824, 2825, 2826, 282
7, 2828, 2829, 2830, 2831, 2832, 2833, 2834, 2835, 2836, 2837, 2838, 2839, 2840, 2
841, 2842, 2843, 2844, 2845, 2846, 2847, 2848, 2849, 2850, 2851, 2852, 2853, 2854,
2855, 2856, 2857, 2858, 2859, 2860, 2861, 2862, 2863, 2864, 2865, 2866, 2867, 286
8, 2869, 2870, 2871, 2872, 2873, 2874, 2875, 2876, 2877, 2878, 2879, 2880, 2881, 2
882, 2883, 2884, 2885, 2886, 2887, 2888, 2889, 2890, 2891, 2892, 2893, 2894, 2895,
2896, 2897, 2898, 2899, 2900, 2901, 2902, 2903, 2904, 2905, 2906, 2907, 2908, 290
9, 2910, 2911, 2912, 2913, 2914, 2915, 2916, 2917, 2918, 2919, 2920, 2921, 2922, 2
923, 2924, 2925, 2926, 2927, 2928, 2929, 2930, 2931, 2932, 2933, 2934, 2935, 2936,
2937, 2938, 2939, 2940, 2941, 2942, 2943, 2944, 2945, 2946, 2947, 2948, 2949, 295
0, 2951, 2952, 2953, 2954, 2955, 2956, 2957, 2958, 2959, 2960, 2961, 2962, 2963, 2
964, 2965, 2966, 2967, 2968, 2969, 2970, 2971, 2972, 2973, 2974, 2975, 2976, 2977,
2978, 2979, 2980, 2981, 2982, 2983, 2984, 2985, 2986, 2987, 2988, 2989, 2990, 299
1, 2992, 2993, 2994, 2995, 2996, 2997, 2998, 2999, 3000, 3001, 3002, 3003, 3004, 3
005, 3006, 3007, 3008, 3009, 3010, 3011, 3012, 3013, 3014, 3015, 3016, 3017, 3018,
3019, 3020, 3021, 3022, 3023, 3024, 3025, 3026, 3027, 3028, 3029, 3030, 3031, 303
2, 3033, 3034, 3035, 3036, 3037, 3038, 3039, 3040, 3041, 3042, 3043, 3044, 3045, 3
046, 3047, 3048, 3049, 3050, 3051, 3052, 3053, 3054, 3055, 3056, 3057, 3058, 3059,
3060, 3061, 3062, 3063, 3064, 3065, 3066, 3067, 3068, 3069, 3070, 3071, 3072, 307
3, 3074, 3075, 3076, 3077, 3078, 3079, 3080, 3081, 3082, 3083, 3084, 3085, 3086, 3
087, 3088, 3089, 3090, 3091, 3092, 3093, 3094, 3095, 3096, 3097, 3098, 3099, 3100,
3101, 3102, 3103, 3104, 3105, 3106, 3107, 3108, 3109, 3110, 3111, 3112, 3113, 311
4, 3115, 3116, 3117, 3118, 3119, 3120, 3121, 3122, 3123, 3124, 3125, 3126, 3127, 3
128, 3129, 3130, 3131, 3132, 3133, 3134, 3135, 3136, 3137, 3138, 3139, 3140, 3141,
3142, 3143, 3144, 3145, 3146, 3147, 3148, 3149, 3150, 3151, 3152, 3153, 3154, 315
5, 3156, 3157, 3158, 3159, 3160, 3161, 3162, 3163, 3164, 3165, 3166, 3167, 3168, 3
169, 3170, 3171, 3172, 3173, 3174, 3175, 3176, 3177, 3178, 3179, 3180, 3181, 3182,
3183, 3184, 3185, 3186, 3187, 3188, 3189, 3190, 3191, 3192, 3193, 3194, 3195, 319
6, 3197, 3198, 3199, 3200, 3201, 3202, 3203, 3204, 3205, 3206, 3207, 3208, 3209, 3
210, 3211, 3212, 3213, 3214, 3215, 3216, 3217, 3218, 3219, 3220, 3221, 3222, 3223,
3224, 3225, 3226, 3227, 3228, 3229, 3230, 3231, 3232, 3233, 3234, 3235, 3236, 323
7, 3238, 3239, 3240, 3241, 3242, 3243, 3244, 3245, 3246, 3247, 3248, 3249, 3250, 3
251, 3252, 3253, 3254, 3255, 3256, 3257, 3258, 3259, 3260, 3261, 3262, 3263, 3264,
3265, 3266, 3267, 3268, 3269, 3270, 3271, 3272, 3273, 3274, 3275, 3276, 3277, 327
8, 3279, 3280, 3281, 3282, 3283, 3284, 3285, 3286, 3287, 3288, 3289, 3290, 3291, 3
292, 3293, 3294, 3295, 3296, 3297, 3298, 3299, 3300, 3301, 3302, 3303, 3304, 3305,
3306, 3307, 3308, 3309, 3310, 3311, 3312, 3313, 3314, 3315, 3316, 3317, 3318, 331
9, 3320, 3321, 3322, 3323, 3324, 3325, 3326, 3327, 3328, 3329, 3330, 3331, 3332, 3
333, 3334, 3335, 3336, 3337, 3338, 3339, 3340, 3341, 3342, 3343, 3344, 3345, 3346,
3347, 3348, 3349, 3350, 3351, 3352, 3353, 3354, 3355, 3356, 3357, 3358, 3359, 336
0, 3361, 3362, 3363, 3364, 3365, 3366, 3367, 3368, 3369, 3370, 3371, 3372, 3373, 3
374, 3375, 3376, 3377, 3378, 3379, 3380, 3381, 3382, 3383, 3384, 3385, 3386, 3387,
3388, 3389, 3390, 3391, 3392, 3393, 3394, 3395, 3396, 3397, 3398, 3399, 3400, 340
1, 3402, 3403, 3404, 3405, 3406, 3407, 3408, 3409, 3410, 3411, 3412, 3413, 3414, 3
415, 3416, 3417, 3418, 3419, 3420, 3421, 3422, 3423, 3424, 3425, 3426, 3427, 3428,
3429, 3430, 3431, 3432, 3433, 3434, 3435, 3436, 3437, 3438, 3439, 3440, 3441, 344
2, 3443, 3444, 3445, 3446, 3447, 3448, 3449, 3450, 3451, 3452, 3453, 3454, 3455, 3
456, 3457, 3458, 3459, 3460, 3461, 3462, 3463, 3464, 3465, 3466, 3467, 3468, 3469,
3470, 3471, 3472, 3473, 3474, 3475, 3476, 3477, 3478, 3479, 3480, 3481, 3482, 348
3, 3484, 3485, 3486, 3487, 3488, 3489, 3490, 3491, 3492, 3493, 3494, 3495, 3496, 3
497, 3498, 3499, 3500, 3501, 3502, 3503, 3504, 3505, 3506, 3507, 3508, 3509, 3510,
3511, 3512, 3513, 3514, 3515, 3516, 3517, 3518, 3519, 3520, 3521, 3522, 3523, 352
4, 3525, 3526, 3527, 3528, 3529, 3530, 3531, 3532, 3533, 3534, 3535, 3536, 3537, 3
538, 3539, 3540, 3541, 3542, 3543, 3544, 3545, 3546, 3547, 3548, 3549, 3550, 3551,
3552, 3553, 3554, 3555, 3556, 3557, 3558, 3559, 3560, 3561, 3562, 3563, 3564, 356
5, 3566, 3567, 3568, 3569, 3570, 3571, 3572, 3573, 3574, 3575, 3576, 3577, 3578, 3
579, 3580, 3581, 3582, 3583, 3584, 3585, 3586, 3587, 3588, 3589, 3590, 3591, 3592,
3593, 3594, 3595, 3596, 3597, 3598, 3599, 3600, 3601, 3602, 3603, 3604, 3605, 360
6, 3607, 3608, 3609, 3610, 3611, 3612, 3613, 3614, 3615, 3616, 3617, 3618, 3619, 3
620, 3621, 3622, 3623, 3624, 3625, 3626, 3627, 3628, 3629, 3630, 3631, 3632, 3633,
3634, 3635, 3636, 3637, 3638, 3639, 3640, 3641, 3642, 3643, 3644, 3645, 3646, 364
```

```
7, 3648, 3649, 3650, 3651, 3652, 3653, 3654, 3655, 3656, 3657, 3658, 3659, 3660, 3
661, 3662, 3663, 3664, 3665, 3666, 3667, 3668, 3669, 3670, 3671, 3672, 3673, 3674,
3675, 3676, 3677, 3678, 3679, 3680, 3681, 3682, 3683, 3684, 3685, 3686, 3687, 368
8, 3689, 3690, 3691, 3692, 3693, 3694, 3695, 3696, 3697, 3698, 3699, 3700, 3701, 3
702, 3703, 3704, 3705, 3706, 3707, 3708, 3709, 3710, 3711, 3712, 3713, 3714, 3715,
3716, 3717, 3718, 3719, 3720, 3721, 3722, 3723, 3724, 3725, 3726, 3727, 3728, 372
9, 3730, 3731, 3732, 3733, 3734, 3735, 3736, 3737, 3738, 3739, 3740, 3741, 3742, 3
743, 3744, 3745, 3746, 3747, 3748, 3749, 3750, 3751, 3752, 3753, 3754, 3755, 3756,
3757, 3758, 3759, 3760, 3761, 3762, 3763, 3764, 3765, 3766, 3767, 3768, 3769, 377
0, 3771, 3772, 3773, 3774, 3775, 3776, 3777, 3778, 3779, 3780, 3781, 3782, 3783, 3
784, 3785, 3786, 3787, 3788, 3789, 3790, 3791, 3792, 3793, 3794, 3795, 3796, 3797,
3798, 3799, 3800, 3801, 3802, 3803, 3804, 3805, 3806, 3807, 3808, 3809, 3810, 381
1, 3812, 3813, 3814, 3815, 3816, 3817, 3818, 3819, 3820, 3821, 3822, 3823, 3824, 3
825, 3826, 3827, 3828, 3829, 3830, 3831, 3832, 3833, 3834, 3835, 3836, 3837, 3838,
3839, 3840, 3841, 3842, 3843, 3844, 3845, 3846, 3847, 3848, 3849, 3850, 3851, 385
2, 3853, 3854, 3855, 3856, 3857, 3858, 3859, 3860, 3861, 3862, 3863, 3864, 3865, 3
866, 3867, 3868, 3869, 3870, 3871, 3872, 3873, 3874, 3875, 3876, 3877, 3878, 3879,
3880, 3881, 3882, 3883, 3884, 3885, 3886, 3887, 3888, 3889, 3890, 3891, 3892, 389
3, 3894, 3895, 3896, 3897, 3898, 3899, 3900, 3901, 3902, 3903, 3904, 3905, 3906, 3
907, 3908, 3909, 3910, 3911, 3912, 3913, 3914, 3915, 3916, 3917, 3918, 3919, 3920,
3921, 3922, 3923, 3924, 3925, 3926, 3927, 3928, 3929, 3930, 3931, 3932, 3933, 393
4, 3935, 3936, 3937, 3938, 3939, 3940, 3941, 3942, 3943, 3944, 3945, 3946, 3947, 3
948, 3949, 3950, 3951, 3952, 3953, 3954, 3955, 3956, 3957, 3958, 3959, 3960, 3961,
3962, 3963, 3964, 3965, 3966, 3967, 3968, 3969, 3970, 3971, 3972, 3973, 3974, 397
5, 3976, 3977, 3978, 3979, 3980, 3981, 3982, 3983, 3984, 3985, 3986, 3987, 3988, 3
989, 3990, 3991, 3992, 3993, 3994, 3995, 3996, 3997, 3998, 3999, 4000, 4001, 4002,
4003, 4004, 4005, 4006, 4007, 4008, 4009, 4010, 4011, 4012, 4013, 4014, 4015, 401
6, 4017, 4018, 4019, 4020, 4021, 4022, 4023, 4024, 4025, 4026, 4027, 4028, 4029, 4
030, 4031, 4032, 4033, 4034, 4035, 4036, 4037, 4038, 4039, 4040, 4041, 4042, 4043,
4044, 4045, 4046, 4047, 4048, 4049, 4050, 4051, 4052, 4053, 4054, 4055, 4056, 405
7, 4058, 4059, 4060, 4061, 4062, 4063, 4064, 4065, 4066, 4067, 4068, 4069, 4070, 4
071, 4072, 4073, 4074, 4075, 4076, 4077, 4078, 4079, 4080, 4081, 4082, 4083, 4084,
4085, 4086, 4087, 4088, 4089, 4090, 4091, 4092, 4093, 4094, 4095, 4096, 4097, 409
8, 4099, 4100, 4101, 4102, 4103, 4104, 4105, 4106, 4107, 4108, 4109, 4110, 4111, 4
112, 4113, 4114, 4115, 4116, 4117, 4118, 4119, 4120, 4121, 4122, 4123, 4124, 4125,
4126, 4127, 4128, 4129, 4130, 4131, 4132, 4133, 4134, 4135, 4136, 4137, 4138, 413
9, 4140, 4141, 4142, 4143, 4144, 4145, 4146, 4147, 4148, 4149, 4150, 4151, 4152, 4
153, 4154, 4155, 4156, 4157, 4158, 4159, 4160, 4161, 4162, 4163, 4164, 4165, 4166,
4167, 4168, 4169, 4170, 4171, 4172, 4173, 4174, 4175, 4176, 4177, 4178, 4179, 418
0, 4181, 4182, 4183, 4184, 4185, 4186, 4187, 4188, 4189, 4190, 4191, 4192, 4193, 4
194, 4195, 4196, 4197, 4198, 4199, 4200, 4201, 4202, 4203, 4204, 4205, 4206, 4207,
4208, 4209, 4210, 4211, 4212, 4213, 4214, 4215, 4216, 4217, 4218, 4219, 4220, 422
1, 4222, 4223, 4224, 4225, 4226, 4227, 4228, 4229, 4230, 4231, 4232, 4233, 4234, 4
235, 4236, 4237, 4238, 4239, 4240, 4241, 4242, 4243, 4244, 4245, 4246, 4247, 4248,
4249, 4250, 4251, 4252, 4253, 4254, 4255, 4256, 4257, 4258, 4259, 4260, 4261, 426
2, 4263, 4264, 4265, 4266, 4267, 4268, 4269, 4270, 4271, 4272, 4273, 4274, 4275, 4
276, 4277, 4278, 4279, 4280, 4281, 4282, 4283, 4284, 4285, 4286, 4287, 4288, 4289,
4290, 4291, 4292, 4293, 4294, 4295, 4296, 4297, 4298, 4299, 4300, 4301, 4302, 430
3, 4304, 4305, 4306, 4307, 4308, 4309, 4310, 4311, 4312, 4313, 4314, 4315, 4316, 4
317, 4318, 4319, 4320, 4321, 4322, 4323, 4324, 4325, 4326, 4327, 4328, 4329, 4330,
4331, 4332, 4333, 4334, 4335, 4336, 4337, 4338, 4339, 4340, 4341, 4342, 4343, 434
4, 4345, 4346, 4347, 4348, 4349, 4350, 4351, 4352, 4353, 4354, 4355, 4356, 4357, 4
358, 4359, 4360, 4361, 4362, 4363, 4364, 4365, 4366, 4367, 4368, 4369, 4370, 4371,
4372, 4373, 4374, 4375, 4376, 4377, 4378, 4379, 4380, 4381, 4382, 4383, 4384, 438
5, 4386, 4387, 4388, 4389, 4390, 4391, 4392, 4393, 4394, 4395, 4396, 4397, 4398, 4
399, 4400, 4401, 4402, 4403, 4404, 4405, 4406, 4407, 4408, 4409, 4410, 4411, 4412,
4413, 4414, 4415, 4416, 4417, 4418, 4419, 4420, 4421, 4422, 4423, 4424, 4425, 442
6, 4427, 4428, 4429, 4430, 4431, 4432, 4433, 4434, 4435, 4436, 4437, 4438, 4439, 4
440, 4441, 4442, 4443, 4444, 4445, 4446, 4447, 4448, 4449, 4450, 4451, 4452, 4453,
4454, 4455, 4456, 4457, 4458, 4459, 4460, 4461, 4462, 4463, 4464, 4465, 4466, 446
7, 4468, 4469, 4470, 4471, 4472, 4473, 4474, 4475, 4476, 4477, 4478, 4479, 4480, 4
481, 4482, 4483, 4484, 4485, 4486, 4487, 4488, 4489, 4490, 4491, 4492, 4493, 4494,
4495, 4496, 4497, 4498, 4499, 4500, 4501, 4502, 4503, 4504, 4505, 4506, 4507, 450
8, 4509, 4510, 4511, 4512, 4513, 4514, 4515, 4516, 4517, 4518, 4519, 4520, 4521, 4
```

```
522, 4523, 4524, 4525, 4526, 4527, 4528, 4529, 4530, 4531, 4532, 4533, 4534, 4535,
4536, 4537, 4538, 4539, 4540, 4541, 4542, 4543, 4544, 4545, 4546, 4547, 4548, 454
9, 4550, 4551, 4552, 4553, 4554, 4555, 4556, 4557, 4558, 4559, 4560, 4561, 4562, 4
563, 4564, 4565, 4566, 4567, 4568, 4569, 4570, 4571, 4572, 4573, 4574, 4575, 4576,
4577, 4578, 4579, 4580, 4581, 4582, 4583, 4584, 4585, 4586, 4587, 4588, 4589, 459
0, 4591, 4592, 4593, 4594, 4595, 4596, 4597, 4598, 4599, 4600, 4601, 4602, 4603, 4
604, 4605, 4606, 4607, 4608, 4609, 4610, 4611, 4612, 4613, 4614, 4615, 4616, 4617,
4618, 4619, 4620, 4621, 4622, 4623, 4624, 4625, 4626, 4627, 4628, 4629, 4630, 463
1, 4632, 4633, 4634, 4635, 4636, 4637, 4638, 4639, 4640, 4641, 4642, 4643, 4644, 4
645, 4646, 4647, 4648, 4649, 4650, 4651, 4652, 4653, 4654, 4655, 4656, 4657, 4658,
4659, 4660, 4661, 4662, 4663, 4664, 4665, 4666, 4667, 4668, 4669, 4670, 4671, 467
2, 4673, 4674, 4675, 4676, 4677, 4678, 4679, 4680, 4681, 4682, 4683, 4684, 4685, 4
686, 4687, 4688, 4689, 4690, 4691, 4692, 4693, 4694, 4695, 4696, 4697, 4698, 4699,
4700, 4701, 4702, 4703, 4704, 4705, 4706, 4707, 4708, 4709, 4710, 4711, 4712, 471
3, 4714, 4715, 4716, 4717, 4718, 4719, 4720, 4721, 4722, 4723, 4724, 4725, 4726, 4
727, 4728, 4729, 4730, 4731, 4732, 4733, 4734, 4735, 4736, 4737, 4738, 4739, 4740,
4741, 4742, 4743, 4744, 4745, 4746, 4747, 4748, 4749, 4750, 4751, 4752, 4753, 475
4, 4755, 4756, 4757, 4758, 4759, 4760, 4761, 4762, 4763, 4764, 4765, 4766, 4767, 4
768, 4769, 4770, 4771, 4772, 4773, 4774, 4775, 4776, 4777, 4778, 4779, 4780, 4781,
4782, 4783, 4784, 4785, 4786, 4787, 4788, 4789, 4790, 4791, 4792, 4793, 4794, 479
5, 4796, 4797, 4798, 4799, 4800, 4801, 4802, 4803, 4804, 4805, 4806, 4807, 4808, 4
809, 4810, 4811, 4812, 4813, 4814, 4815, 4816, 4817, 4818, 4819, 4820, 4821, 4822,
4823, 4824, 4825, 4826, 4827, 4828, 4829, 4830, 4831, 4832, 4833, 4834, 4835, 483
6, 4837, 4838, 4839, 4840, 4841, 4842, 4843, 4844, 4845, 4846, 4847, 4848, 4849, 4
850, 4851, 4852, 4853, 4854, 4855, 4856, 4857, 4858, 4859, 4860, 4861, 4862, 4863,
4864, 4865, 4866, 4867, 4868, 4869, 4870, 4871, 4872, 4873, 4874, 4875, 4876, 487
7, 4878, 4879, 4880, 4881, 4882, 4883, 4884, 4885, 4886, 4887, 4888, 4889, 4890, 4
891, 4892, 4893, 4894, 4895, 4896, 4897, 4898, 4899, 4900, 4901, 4902, 4903, 4904,
4905, 4906, 4907, 4908, 4909, 4910, 4911, 4912, 4913, 4914, 4915, 4916, 4917, 491
8, 4919, 4920, 4921, 4922, 4923, 4924, 4925, 4926, 4927, 4928, 4929, 4930, 4931, 4
932, 4933, 4934, 4935, 4936, 4937, 4938, 4939, 4940, 4941, 4942, 4943, 4944, 4945,
4946, 4947, 4948, 4949, 4950, 4951, 4952, 4953, 4954, 4955, 4956, 4957, 4958, 495
9, 4960, 4961, 4962, 4963, 4964, 4965, 4966, 4967, 4968, 4969, 4970, 4971, 4972, 4
973, 4974, 4975, 4976, 4977, 4978, 4979, 4980, 4981, 4982, 4983, 4984, 4985, 4986,
4987, 4988, 4989, 4990, 4991, 4992, 4993, 4994, 4995, 4996, 4997, 4998, 4999, 500
0}

******************************************************************* Age **********
**************************************************************

{23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 4
3, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
64, 65, 66, 67}

******************************************************************* Experience ***
*********************************************************************

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 4
3, -2, -3, -1}

******************************************************************* Income *******
*****************************************************************

{8, 9, 10, 11, 12, 13, 14, 15, 18, 19, 20, 21, 22, 23, 24, 25, 28, 29, 30, 31, 32,
33, 34, 35, 38, 39, 40, 41, 42, 43, 44, 45, 48, 49, 50, 51, 52, 53, 54, 55, 58, 5
9, 60, 61, 62, 63, 64, 65, 68, 69, 70, 71, 72, 73, 74, 75, 78, 79, 80, 81, 82, 83,
84, 85, 88, 89, 90, 91, 92, 93, 94, 95, 98, 99, 100, 101, 102, 103, 104, 105, 108,
109, 110, 111, 112, 113, 114, 115, 118, 119, 120, 121, 122, 123, 124, 125, 128, 12
9, 130, 131, 132, 133, 134, 135, 138, 139, 140, 141, 142, 143, 144, 145, 148, 149,
150, 151, 152, 153, 154, 155, 158, 159, 160, 161, 162, 163, 164, 165, 168, 169, 17
0, 171, 172, 173, 174, 175, 178, 179, 180, 181, 182, 183, 184, 185, 188, 189, 190,
191, 192, 193, 194, 195, 198, 199, 200, 201, 202, 203, 204, 205, 218, 224}
```

```
****************************************************************** ZIP Code *****
******************************************************************

{92161, 92173, 92177, 92182, 94234, 92192, 92220, 92251, 94301, 94302, 94303, 9430
4, 94305, 94306, 90210, 90212, 94309, 90230, 90232, 90245, 90250, 90254, 90266, 90
272, 90274, 90275, 90277, 92325, 90280, 92333, 90291, 92346, 92350, 90304, 92354,
94402, 94404, 92373, 92374, 92399, 92407, 90011, 90016, 90401, 90404, 94501, 9040
5, 90018, 92054, 94507, 94509, 94521, 94523, 94526, 94534, 94536, 94538, 94539, 94
542, 94545, 94546, 94550, 94551, 94553, 92507, 94555, 92064, 94558, 92518, 94566,
92521, 94571, 94575, 94577, 94583, 94588, 94590, 94591, 94596, 90502, 90503, 9050
4, 90505, 94108, 96651, 94604, 90509, 94606, 94607, 94608, 94609, 94610, 94611, 94
612, 94618, 94116, 92606, 92612, 92614, 92624, 92626, 92630, 92634, 94124, 92646,
92647, 92648, 90601, 92653, 94701, 94703, 94704, 94705, 94706, 94707, 94708, 9470
9, 94710, 92661, 92660, 90623, 94720, 92672, 92673, 92675, 92677, 90630, 90638, 90
639, 90640, 92691, 92692, 92694, 92697, 90650, 92703, 92704, 92705, 92709, 92717,
92735, 94801, 94803, 94806, 90717, 90720, 92780, 90740, 90745, 90747, 90755, 9280
6, 92807, 92821, 92831, 92833, 92834, 92835, 92843, 94901, 94904, 90813, 92866, 92
867, 92868, 92870, 94920, 94923, 94928, 92886, 90840, 94939, 94949, 94960, 94965,
94970, 94998, 95003, 95005, 95006, 95008, 95010, 95014, 94598, 95020, 95023, 9503
2, 95035, 95037, 95039, 95045, 95051, 93003, 95053, 95054, 93009, 93010, 95060, 93
014, 95064, 93022, 93023, 95070, 93033, 91006, 91007, 93063, 91016, 95112, 93065,
95120, 91024, 95123, 95125, 95126, 91030, 95131, 95133, 95134, 95135, 95136, 9104
0, 95138, 93101, 93105, 93106, 93107, 93108, 93109, 93111, 93117, 93118, 95192, 95
193, 91101, 91103, 91105, 91107, 91109, 95207, 95211, 91116, 91125, 91129, 91203,
91207, 95307, 9307, 95348, 93302, 95351, 93305, 95354, 93311, 95370, 91301, 91302,
91304, 95403, 95405, 91311, 91320, 91326, 95422, 91330, 91335, 91342, 91343, 9134
5, 93401, 95449, 91355, 93403, 93407, 91360, 91361, 91365, 91367, 91380, 95482, 93
437, 91401, 95503, 93460, 95518, 91423, 95521, 93524, 93555, 95605, 93561, 95616,
95617, 95621, 95630, 93611, 95670, 95678, 91604, 91605, 93657, 91614, 95741, 9574
7, 95758, 93711, 95762, 93720, 93727, 91706, 91709, 91710, 91711, 95812, 95814, 95
816, 95817, 95818, 95819, 95820, 95821, 95822, 95825, 91730, 95827, 95828, 95831,
95833, 91741, 91745, 95841, 95842, 91754, 91763, 91765, 91768, 91770, 91773, 9177
5, 91784, 91791, 91801, 95929, 93907, 95973, 93933, 93940, 93943, 91902, 93950, 96
001, 93955, 96003, 91910, 91911, 96008, 91941, 91942, 91950, 94002, 94005, 94010,
94015, 96064, 94019, 94022, 94024, 94025, 94028, 94035, 94040, 94043, 96091, 9609
4, 92007, 92008, 92009, 94061, 94063, 94065, 94066, 92024, 92028, 92029, 94080, 92
037, 94085, 92038, 94086, 94087, 96145, 90005, 94102, 90007, 92056, 90009, 94104,
94105, 96150, 94109, 94107, 94111, 94110, 94112, 94114, 94115, 90019, 94117, 9411
8, 92069, 90024, 92068, 94122, 94123, 90028, 90029, 90027, 90025, 94126, 90033, 90
034, 90035, 90036, 94132, 90037, 90032, 94131, 90041, 92084, 90044, 92093, 90045,
94143, 92096, 90049, 90048, 92101, 92103, 92104, 90057, 90058, 92106, 90059, 9210
9, 92110, 90064, 90065, 90066, 92115, 92116, 90068, 90071, 92120, 92121, 92122, 90
073, 92123, 92124, 92126, 92129, 92130, 92131, 90086, 90089, 90095, 92152, 92154}

****************************************************************** Family *******
****************************************************************

{1, 2, 3, 4}

****************************************************************** CCAvg ********
***************************************************************

{'1/30', '1/10', '0/40', '1/00', '2/33', '1/67', '5/90', '6/33', '0/10', '5/80',
'0/67', '5/10', '5/50', '3/70', '6/10', '3/20', '4/70', '7/60', '6/60', '3/40',
'4/90', '6/50', '8/80', '3/33', '3/10', '1/90', '8/20', '4/10', '2/80', '7/00',
'8/30', '2/75', '1/60', '4/75', '0/90', '2/20', '3/60', '10/00', '5/20', '6/67',
'4/33', '4/00', '9/30', '4/40', '4/80', '4/60', '1/75', '9/00', '7/30', '4/30',
'5/33', '1/33', '2/10', '6/20', '3/80', '8/10', '7/50', '2/70', '6/80', '2/50',
'1/80', '0/60', '1/70', '6/00', '8/60', '4/20', '5/67', '2/90', '5/40', '5/70',
'5/60', '6/70', '7/40', '7/80', '0/20', '3/25', '3/30', '0/00', '5/00', '4/50',
'3/67', '4/25', '4/67', '8/90', '0/75', '1/50', '6/90', '7/20', '2/60', '8/00',
'2/40', '6/40', '2/67', '3/50', '0/80', '3/00', '0/50', '1/20', '8/50', '7/90',
'2/00', '0/70', '0/30', '3/90', '5/30', '2/30', '6/30', '1/40'}
```

```
****************************************************************** Education ****
*******************************************************************

{1, 2, 3}

****************************************************************** Mortgage *****
******************************************************************

{0, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 9
4, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,
112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 12
8, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144,
145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 16
1, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177,
178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 19
4, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210,
211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 22
7, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243,
244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 255, 256, 257, 258, 259, 260, 26
2, 263, 264, 265, 266, 267, 268, 270, 271, 272, 273, 275, 276, 277, 278, 280, 281,
282, 283, 284, 285, 286, 287, 289, 290, 292, 293, 294, 295, 296, 297, 298, 299, 30
0, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 318,
319, 321, 322, 323, 325, 326, 327, 328, 329, 330, 331, 333, 334, 336, 337, 341, 34
2, 343, 344, 345, 351, 352, 353, 354, 355, 357, 358, 359, 360, 361, 364, 366, 368,
372, 373, 374, 378, 380, 381, 382, 383, 385, 389, 391, 392, 394, 396, 397, 398, 40
0, 402, 403, 405, 406, 408, 410, 412, 415, 416, 419, 421, 422, 427, 428, 429, 431,
432, 433, 437, 442, 446, 449, 452, 455, 458, 461, 464, 466, 467, 470, 475, 477, 48
1, 483, 485, 496, 500, 505, 508, 509, 522, 524, 535, 541, 547, 550, 553, 565, 567,
569, 571, 577, 581, 582, 587, 589, 590, 601, 612, 617, 635}

****************************************************************** Personal Loan
**********************************************************************

{0, 1}

****************************************************************** Securities Acc
ount **********************************************************************

{0, 1}

****************************************************************** CD Account ***
********************************************************************

{0, 1}

****************************************************************** Online *******
****************************************************************

{0, 1}

****************************************************************** CreditCard ***
********************************************************************

{0, 1}
```

In [10]:

```python
#Checking column names
df.columns
```

Out[10]:

```
Index(['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg',
       'Education', 'Mortgage', 'Personal Loan', 'Securities Account',
       'CD Account', 'Online', 'CreditCard'],
      dtype='object')
```

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   ID                  5000 non-null   int64 
 1   Age                 5000 non-null   int64 
 2   Experience          5000 non-null   int64 
 3   Income              5000 non-null   int64 
 4   ZIP Code            5000 non-null   int64 
 5   Family              5000 non-null   int64 
 6   CCAvg               5000 non-null   object
 7   Education           5000 non-null   int64 
 8   Mortgage            5000 non-null   int64 
 9   Personal Loan       5000 non-null   int64 
 10  Securities Account  5000 non-null   int64 
 11  CD Account          5000 non-null   int64 
 12  Online              5000 non-null   int64 
 13  CreditCard          5000 non-null   int64 
dtypes: int64(13), object(1)
memory usage: 547.0+ KB
```

In [12]:
```
df['Income'] = df['Income'] / 12
```

In [13]:
```
# Replace negative values with NaN
df['Experience'] = np.where(df['Experience'] < 0, np.nan, df['Experience'])

# Impute NaN values with the mean or median
df['Experience'].fillna(df['Experience'].mean(), inplace=True)
```

In [14]:
```
df.isnull().sum()
```

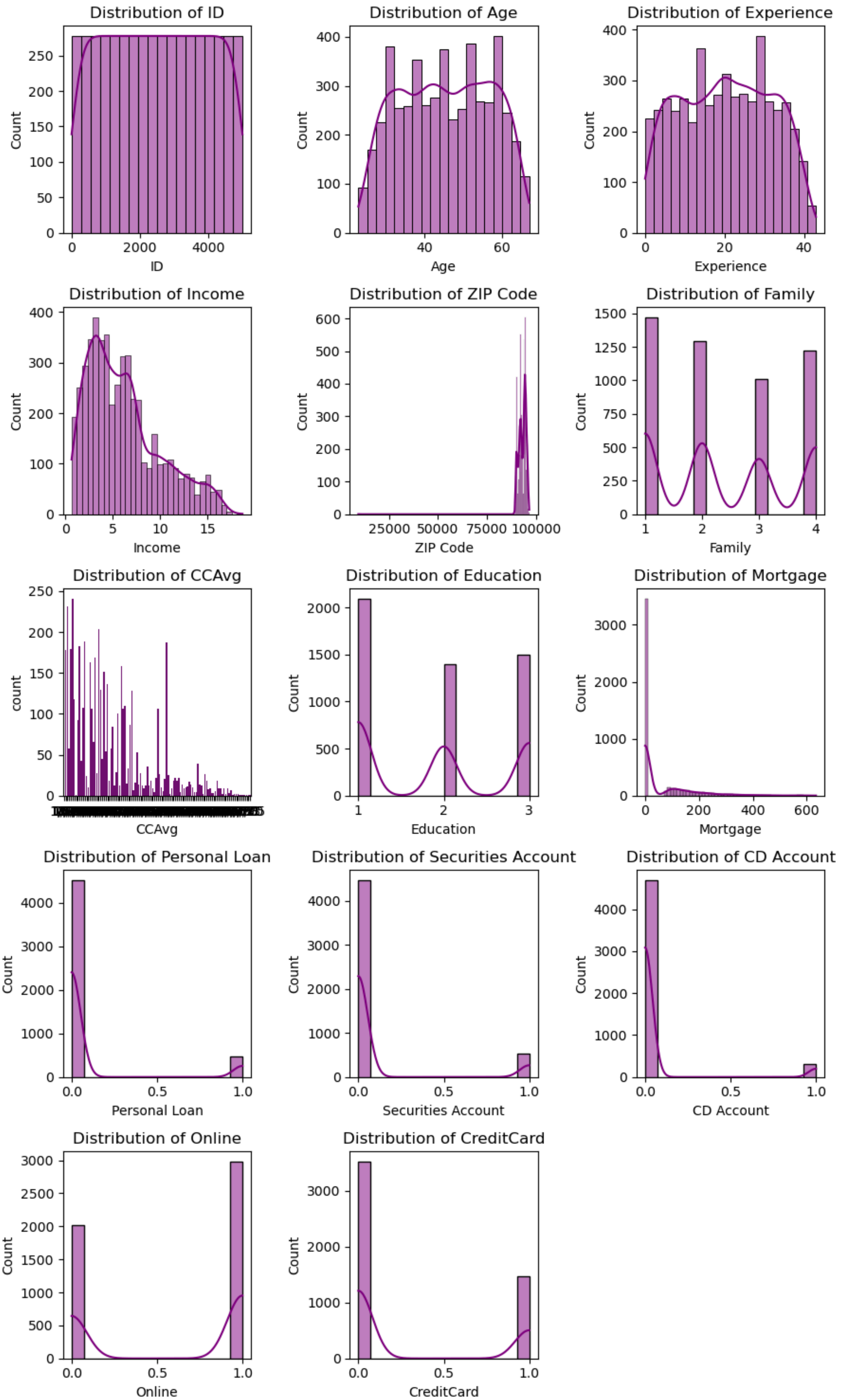
Out[14]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

In [15]:
```
#Remove rows with Negative values

df = df[df['Experience'] >= 0]
```

# EXPLORATORY DATA ANALYSIS(EDA)

In [16]:
```
# Plotting distribution of each column in the dataset
# Defining the number of rows and columns for the grid
num_cols = 3
num_rows = (len(df.columns) + num_cols - 1) // num_cols
# Creating a grid of subplots
```

```
fig, axes = plt.subplots(num_rows, num_cols, figsize=(3*num_cols, 3*num_rows))
# Flatten the axes array
axes = axes.flatten()
# Iterating over each column in the DataFrame
for i, column in enumerate(df.columns):
    ax = axes[i]  # Get the subplot axes
    if df[column].dtype == 'object':
        # Categorical variable, plot count plot
        sns.countplot(x=column, data=df, ax=ax, color='purple')
    else:
        # Numerical variable, plot histogram
        sns.histplot(x=column, data=df, kde=True, ax=ax, color='purple')
    ax.set_title(f"Distribution of {column}")
# Remove empty subplots
for i in range(len(df.columns), num_rows * num_cols):
    fig.delaxes(axes[i])
# Adjust layout
plt.tight_layout()
# save image to include in pdf
plt.savefig('Distribution of each Feature.jpg')
plt.show()
```




# UNIVARIATE ANALYSIS

In [17]:
```
numeric_stats = df.describe()
print(numeric_stats)
```

```
                ID          Age   Experience       Income      ZIP Code  \
count  5000.000000  5000.000000  5000.000000  5000.000000   5000.000000
mean   2500.500000    45.338400    20.331043     6.147850  93152.503000
std    1443.520003    11.463166    11.252985     3.836144   2121.852197
min       1.000000    23.000000     0.000000     0.666667   9307.000000
25%    1250.750000    35.000000    11.000000     3.250000  91911.000000
50%    2500.500000    45.000000    20.331043     5.333333  93437.000000
75%    3750.250000    55.000000    30.000000     8.166667  94608.000000
max    5000.000000    67.000000    43.000000    18.666667  96651.000000

            Family    Education     Mortgage  Personal Loan  \
count  5000.000000  5000.000000  5000.000000    5000.000000
mean      2.396400     1.881000    56.498800       0.096000
std       1.147663     0.839869   101.713802       0.294621
min       1.000000     1.000000     0.000000       0.000000
25%       1.000000     1.000000     0.000000       0.000000
50%       2.000000     2.000000     0.000000       0.000000
75%       3.000000     3.000000   101.000000       0.000000
max       4.000000     3.000000   635.000000       1.000000

       Securities Account  CD Account       Online   CreditCard
count         5000.000000  5000.00000  5000.000000  5000.000000
mean             0.104400     0.06040     0.596800     0.294000
std              0.305809     0.23825     0.490589     0.455637
min              0.000000     0.00000     0.000000     0.000000
25%              0.000000     0.00000     0.000000     0.000000
50%              0.000000     0.00000     1.000000     0.000000
75%              0.000000     0.00000     1.000000     1.000000
max              1.000000     1.00000     1.000000     1.000000
```

In [18]:
```
categorical_features = df.drop(columns=['Age','Income','Experience','CCAvg','Mortga
categorical_features
```

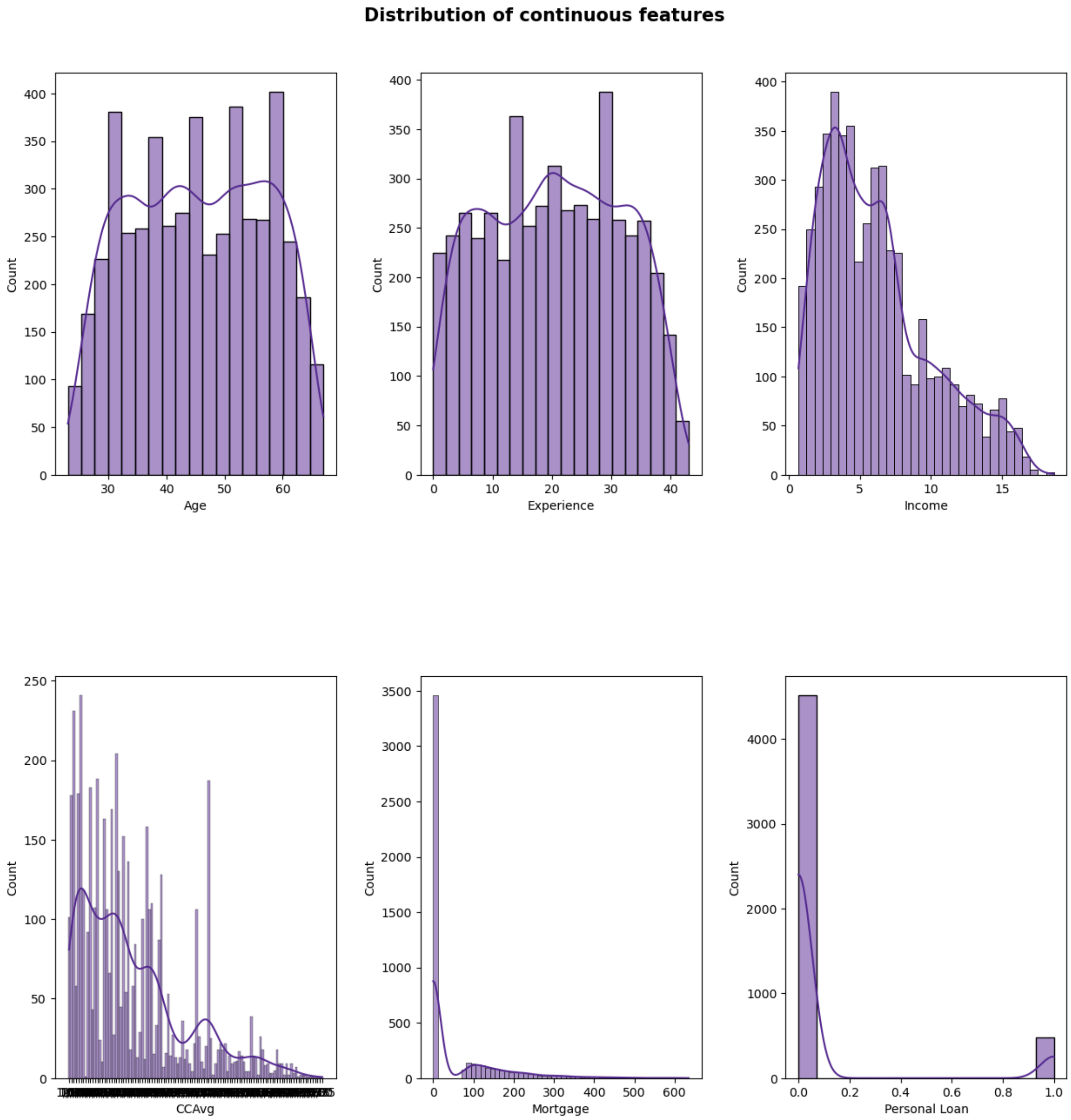
Out[18]:

| | ID | ZIP Code | Family | Education | Personal Loan | Securities Account | CD Account | Online | CreditCard |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 91107 | 4 | 1 | 0 | 1 | 0 | 0 | 0 |
| **1** | 2 | 90089 | 3 | 1 | 0 | 1 | 0 | 0 | 0 |
| **2** | 3 | 94720 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| **3** | 4 | 94112 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| **4** | 5 | 91330 | 4 | 2 | 0 | 0 | 0 | 0 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **4995** | 4996 | 92697 | 1 | 3 | 0 | 0 | 0 | 1 | 0 |
| **4996** | 4997 | 92037 | 4 | 1 | 0 | 0 | 0 | 1 | 0 |
| **4997** | 4998 | 93023 | 2 | 3 | 0 | 0 | 0 | 0 | 0 |
| **4998** | 4999 | 90034 | 3 | 2 | 0 | 0 | 0 | 1 | 0 |
| **4999** | 5000 | 92612 | 3 | 1 | 0 | 0 | 0 | 1 | 1 |

5000 rows × 9 columns

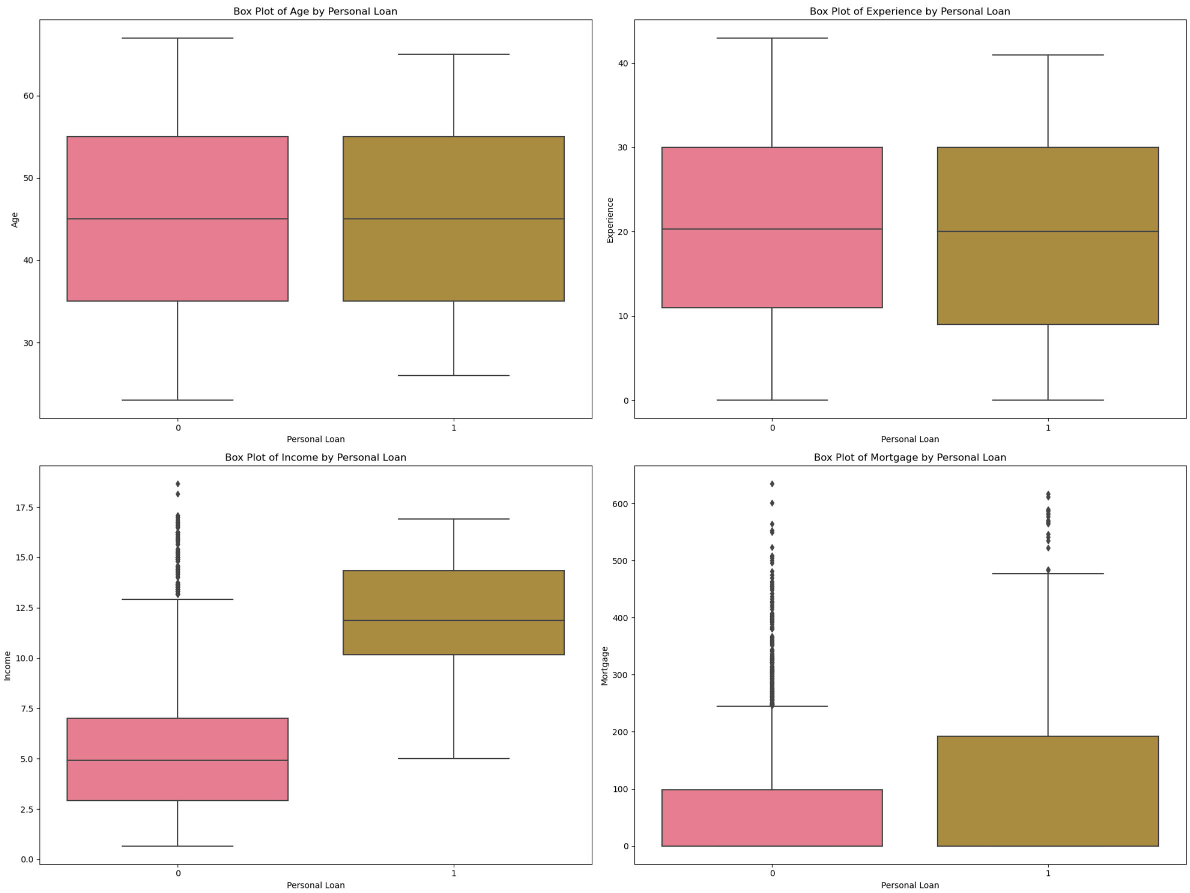
In [19]:
```
continuous_features = df.drop(columns=['ZIP Code','Family','Education','Securities
                                       ,'Online','CreditCard', 'ID'])
```

In [20]:
```
sns.set_palette("Purples_r")  # Use "Purples" for regular order, "Purples_r" for re
fig, ax = plt.subplots(2, 3, figsize=(15, 15), gridspec_kw={"hspace": 0.5, "wspace'

for i, col in enumerate(continuous_features):
    sns.histplot(df[col], kde=True, ax=ax[i//3, i%3])

fig.suptitle('Distribution of continuous features', fontsize=15, fontweight='bold',
plt.show()
```




In [21]:
```
purple_palette = sns.color_palette("husl")

# Create a single figure to hold all the subplots
fig, axes = plt.subplots(2, 2, figsize=(20, 15))

# Plot Age vs Personal Loan
sns.boxplot(ax=axes[0, 0], x="Personal Loan", y="Age", data=df, palette=purple_pale
axes[0, 0].set_title('Box Plot of Age by Personal Loan')

# Plot Experience vs Personal Loan
sns.boxplot(ax=axes[0, 1], x="Personal Loan", y="Experience", data=df, palette=purp
axes[0, 1].set_title('Box Plot of Experience by Personal Loan')

# Plot Income vs Personal Loan
```

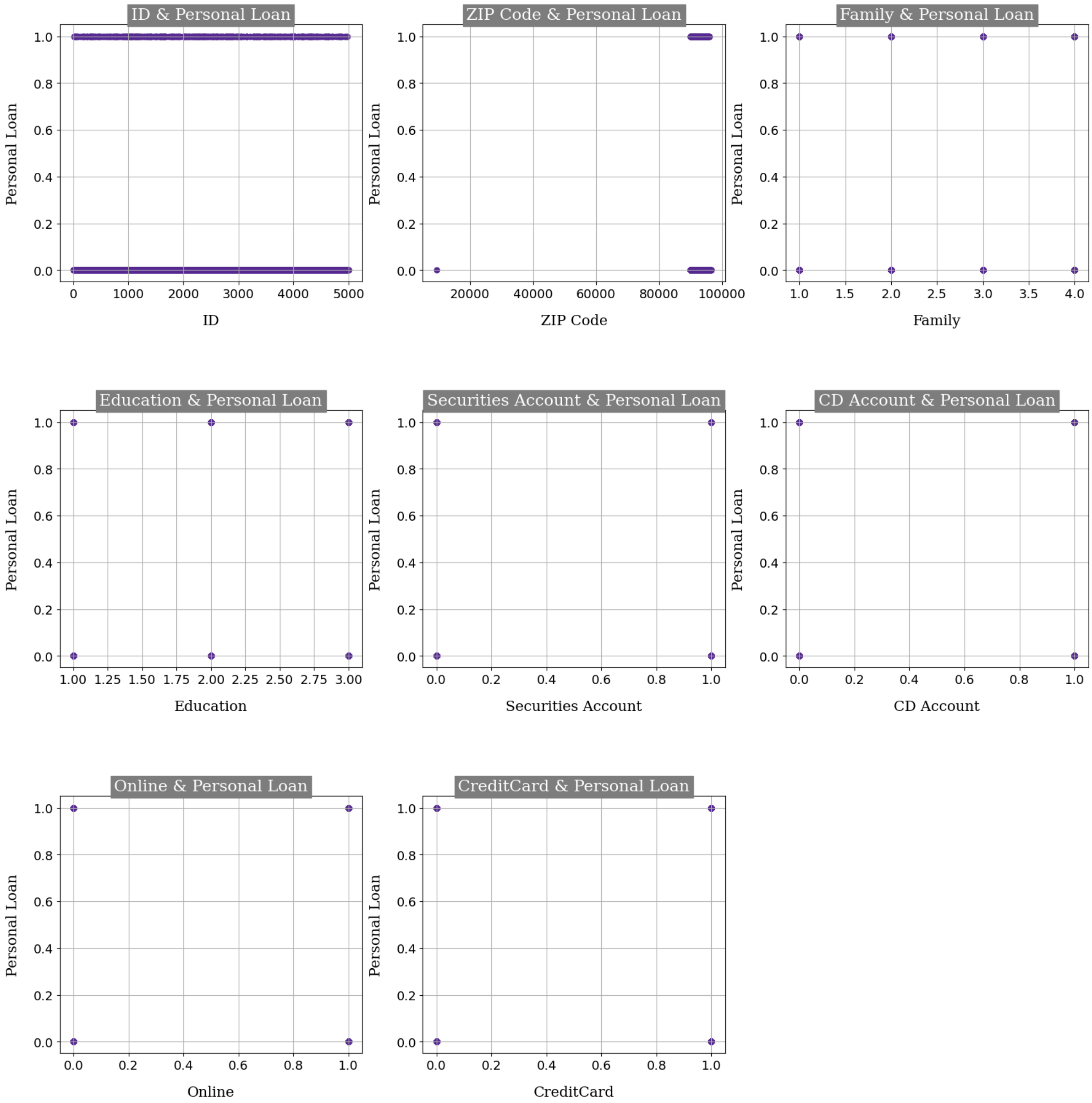
```
sns.boxplot(ax=axes[1, 0], x="Personal Loan", y="Income", data=df, palette=purple_p
axes[1, 0].set_title('Box Plot of Income by Personal Loan')

# Plot Mortgage vs Personal Loan
sns.boxplot(ax=axes[1, 1], x="Personal Loan", y="Mortgage", data=df, palette=purple
axes[1, 1].set_title('Box Plot of Mortgage by Personal Loan')

# Adjust layout
plt.tight_layout()

plt.savefig('BoxPlots_All_Variables.jpg')

# Show all the plots
plt.show()
```




In [22]:

```
font1 = {'family': 'serif', 'size': 18}
font2 = {'family': 'serif', 'size': 16}

f = plt.figure()
f.set_figwidth(20)
f.set_figheight(20)

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.2,
                    hspace=0.5)

i = 0
for column in categorical_features.drop(columns=['Personal Loan']):
    i += 1
    plt.subplot(3, 3, i)
    plt.scatter(categorical_features[column], categorical_features["Personal Loan"]
    plt.title(column + " & Personal Loan", backgroundcolor='grey', color='white', f
    plt.xticks(fontsize=14)
```

```
    plt.yticks(fontsize=14)
    plt.xlabel(column, fontdict=font2, labelpad=15)
    plt.ylabel("Personal Loan", fontdict=font2, labelpad=15)
    plt.grid()

plt.show()
```




In [23]:
```
#Drop the id column

df.drop(['ID', 'ZIP Code'],axis=1,inplace=True)
df
```

Out[23]:

| | Age | Experience | Income | Family | CCAvg | Education | Mortgage | Personal Loan | Securities Account | Acc |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 25 | 1.0 | 4.083333 | 4 | 1/60 | 1 | 0 | 0 | 1 | |
| **1** | 45 | 19.0 | 2.833333 | 3 | 1/50 | 1 | 0 | 0 | 1 | |
| **2** | 39 | 15.0 | 0.916667 | 1 | 1/00 | 1 | 0 | 0 | 0 | |
| **3** | 35 | 9.0 | 8.333333 | 1 | 2/70 | 2 | 0 | 0 | 0 | |
| **4** | 35 | 8.0 | 3.750000 | 4 | 1/00 | 2 | 0 | 0 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **4995** | 29 | 3.0 | 3.333333 | 1 | 1/90 | 3 | 0 | 0 | 0 | |
| **4996** | 30 | 4.0 | 1.250000 | 4 | 0/40 | 1 | 85 | 0 | 0 | |
| **4997** | 63 | 39.0 | 2.000000 | 2 | 0/30 | 3 | 0 | 0 | 0 | |
| **4998** | 65 | 40.0 | 4.083333 | 3 | 0/50 | 2 | 0 | 0 | 0 | |
| **4999** | 28 | 4.0 | 6.916667 | 3 | 0/80 | 1 | 0 | 0 | 0 | |

5000 rows × 12 columns

In [24]:
```
print(df['CCAvg'].unique())
```

```
['1/60' '1/50' '1/00' '2/70' '0/40' '0/30' '0/60' '8/90' '2/40' '0/10'
 '3/80' '2/50' '2/00' '4/70' '8/10' '0/50' '0/90' '1/20' '0/70' '3/90'
 '0/20' '2/20' '3/30' '1/80' '2/90' '1/40' '5/00' '2/30' '1/10' '5/70'
 '4/50' '2/10' '8/00' '1/70' '0/00' '2/80' '3/50' '4/00' '2/60' '1/30'
 '5/60' '5/20' '3/00' '4/60' '3/60' '7/20' '1/75' '7/40' '2/67' '7/50'
 '6/50' '7/80' '7/90' '4/10' '1/90' '4/30' '6/80' '5/10' '3/10' '0/80'
 '3/70' '6/20' '0/75' '2/33' '4/90' '0/67' '3/20' '5/50' '6/90' '4/33'
 '7/30' '4/20' '4/40' '6/10' '6/33' '6/60' '5/30' '3/40' '7/00' '6/30'
 '8/30' '6/00' '1/67' '8/60' '7/60' '6/40' '10/00' '5/90' '5/40' '8/80'
 '1/33' '9/00' '6/70' '4/25' '6/67' '5/80' '4/80' '3/25' '5/67' '8/50'
 '4/75' '4/67' '3/67' '8/20' '3/33' '5/33' '9/30' '2/75']
```

In [25]:
```
from fractions import Fraction
df['CCAvg'] = df['CCAvg'].apply(lambda x: float(Fraction(x.replace('/', '.'))))
df['CCAvg'].fillna(df['CCAvg'].mean(), inplace=True)
```

In [26]:
```
print(df['CCAvg'].unique())
```

```
[ 1.6   1.5   1.    2.7   0.4   0.3   0.6   8.9   2.4   0.1   3.8   2.5
  2.    4.7   8.1   0.5   0.9   1.2   0.7   3.9   0.2   2.2   3.3   1.8
  2.9   1.4   5.    2.3   1.1   5.7   4.5   2.1   8.    1.7   0.    2.8
  3.5   4.    2.6   1.3   5.6   5.2   3.    4.6   3.6   7.2   1.75  7.4
  2.67  7.5   6.5   7.8   7.9   4.1   1.9   4.3   6.8   5.1   3.1   0.8
  3.7   6.2   0.75  2.33  4.9   0.67  3.2   5.5   6.9   4.33  7.3   4.2
  4.4   6.1   6.33  6.6   5.3   3.4   7.    6.3   8.3   6.    1.67  8.6
  7.6   6.4  10.    5.9   5.4   8.8   1.33  9.    6.7   4.25  6.67  5.8
  4.8   3.25  5.67  8.5   4.75  4.67  3.67  8.2   3.33  5.33  9.3   2.75]
```

In [27]:
```
# Printing head of dataset after cleaning
df.head(n=20)
```

Out[27]:

| | Age | Experience | Income | Family | CCAvg | Education | Mortgage | Personal Loan | Securities Account | Accou |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 25 | 1.0 | 4.083333 | 4 | 1.6 | 1 | 0 | 0 | 1 | |
| **1** | 45 | 19.0 | 2.833333 | 3 | 1.5 | 1 | 0 | 0 | 1 | |
| **2** | 39 | 15.0 | 0.916667 | 1 | 1.0 | 1 | 0 | 0 | 0 | |
| **3** | 35 | 9.0 | 8.333333 | 1 | 2.7 | 2 | 0 | 0 | 0 | |
| **4** | 35 | 8.0 | 3.750000 | 4 | 1.0 | 2 | 0 | 0 | 0 | |
| **5** | 37 | 13.0 | 2.416667 | 4 | 0.4 | 2 | 155 | 0 | 0 | |
| **6** | 53 | 27.0 | 6.000000 | 2 | 1.5 | 2 | 0 | 0 | 0 | |
| **7** | 50 | 24.0 | 1.833333 | 1 | 0.3 | 3 | 0 | 0 | 0 | |
| **8** | 35 | 10.0 | 6.750000 | 3 | 0.6 | 2 | 104 | 0 | 0 | |
| **9** | 34 | 9.0 | 15.000000 | 1 | 8.9 | 3 | 0 | 1 | 0 | |
| **10** | 65 | 39.0 | 8.750000 | 4 | 2.4 | 3 | 0 | 0 | 0 | |
| **11** | 29 | 5.0 | 3.750000 | 3 | 0.1 | 2 | 0 | 0 | 0 | |
| **12** | 48 | 23.0 | 9.500000 | 2 | 3.8 | 3 | 0 | 0 | 1 | |
| **13** | 59 | 32.0 | 3.333333 | 4 | 2.5 | 2 | 0 | 0 | 0 | |
| **14** | 67 | 41.0 | 9.333333 | 1 | 2.0 | 1 | 0 | 0 | 1 | |
| **15** | 60 | 30.0 | 1.833333 | 1 | 1.5 | 3 | 0 | 0 | 0 | |
| **16** | 38 | 14.0 | 10.833333 | 4 | 4.7 | 3 | 134 | 1 | 0 | |
| **17** | 42 | 18.0 | 6.750000 | 4 | 2.4 | 1 | 0 | 0 | 0 | |
| **18** | 46 | 21.0 | 16.083333 | 2 | 8.1 | 3 | 0 | 1 | 0 | |
| **19** | 55 | 28.0 | 1.750000 | 1 | 0.5 | 2 | 0 | 0 | 1 | |

In [28]:

```
df.head(n=20)
df_education = df['Education']
df_education = pd.get_dummies(df_education)
df.drop(['Education'],axis=1,inplace=True)
df[df_education.columns] = df_education
```

In [29]:

```
df[[1,2,3]].rename(columns = {1:'education_1',
                               2:'education_2',
                               3:'education_3'},inplace=True)
```

In [30]:

```
df.dtypes
```

Out[30]:
```
Age                    int64
Experience           float64
Income               float64
Family                 int64
CCAvg                float64
Mortgage               int64
Personal Loan          int64
Securities Account     int64
CD Account             int64
Online                 int64
CreditCard             int64
1                       bool
2                       bool
3                       bool
dtype: object
```

In [31]:
```
# Select boolean columns
boolean_columns = df.select_dtypes(include='bool').columns
# Convert boolean columns to integers (0 and 1)
df[boolean_columns] = df[boolean_columns].astype(int)
```

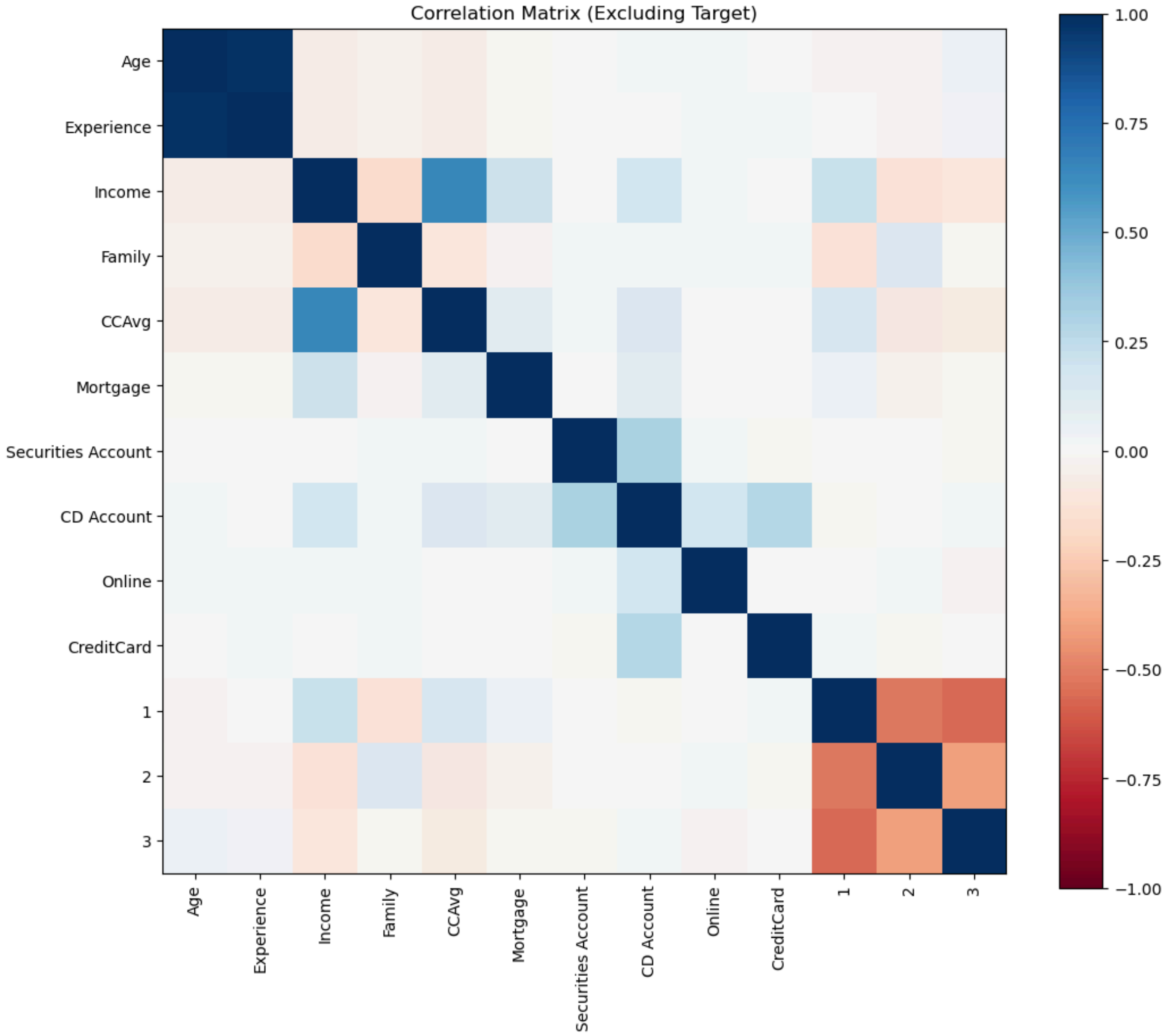
In [32]:
```
df.head()
```

Out[32]:

| | Age | Experience | Income | Family | CCAvg | Mortgage | Personal Loan | Securities Account | CD Account | Online | C |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 25 | 1.0 | 4.083333 | 4 | 1.6 | 0 | 0 | 1 | 0 | 0 | |
| **1** | 45 | 19.0 | 2.833333 | 3 | 1.5 | 0 | 0 | 1 | 0 | 0 | |
| **2** | 39 | 15.0 | 0.916667 | 1 | 1.0 | 0 | 0 | 0 | 0 | 0 | |
| **3** | 35 | 9.0 | 8.333333 | 1 | 2.7 | 0 | 0 | 0 | 0 | 0 | |
| **4** | 35 | 8.0 | 3.750000 | 4 | 1.0 | 0 | 0 | 0 | 0 | 0 | |

# Correllation Analysis

In [33]:
```
features_without_target = df.drop(columns=['Personal Loan'])

# Compute the correlation matrix for features excluding the target
correlation_matrix = features_without_target.corr()

# Plot the correlation matrix using a heatmap
plt.figure(figsize=(12, 10))
plt.imshow(correlation_matrix, cmap='RdBu', vmin= -1, vmax= 1)
plt.colorbar()
plt.title('Correlation Matrix (Excluding Target)')
plt.xticks(ticks=range(len(correlation_matrix.columns)), labels=correlation_matrix.
plt.yticks(ticks=range(len(correlation_matrix.columns)), labels=correlation_matrix.
plt.savefig('Correlation Analysis.jpg')
plt.show()
```




In [34]:
```
print(correlation_matrix)
```

```
                         Age  Experience    Income    Family     CCAvg  \
Age                 1.000000    0.976630 -0.055269 -0.046418 -0.052012
Experience          0.976630    1.000000 -0.049072 -0.045403 -0.048685
Income             -0.055269   -0.049072  1.000000 -0.157501  0.645984
Family             -0.046418   -0.045403 -0.157501  1.000000 -0.109275
CCAvg              -0.052012   -0.048685  0.645984 -0.109275  1.000000
Mortgage           -0.012539   -0.013404  0.206806 -0.020445  0.109905
Securities Account -0.000436   -0.000454 -0.002616  0.019994  0.015086
CD Account          0.008043    0.005449  0.169738  0.014110  0.136534
Online              0.013702    0.013447  0.014206  0.010354 -0.003611
CreditCard          0.007681    0.008830 -0.002385  0.011588 -0.006689
1                  -0.027770   -0.007549  0.218019 -0.118628  0.156979
2                  -0.016264   -0.017334 -0.128364  0.139201 -0.090366
3                   0.045838    0.025119 -0.108878 -0.008744 -0.080413

                    Mortgage  Securities Account  CD Account    Online  \
Age                -0.012539           -0.000436    0.008043  0.013702
Experience         -0.013404           -0.000454    0.005449  0.013447
Income              0.206806           -0.002616    0.169738  0.014206
Family             -0.020445            0.019994    0.014110  0.010354
CCAvg               0.109905            0.015086    0.136534 -0.003611
Mortgage            1.000000           -0.005411    0.089311 -0.005995
Securities Account -0.005411            1.000000    0.317034  0.012627
CD Account          0.089311            0.317034    1.000000  0.175880
Online             -0.005995            0.012627    0.175880  1.000000
CreditCard         -0.007231           -0.015028    0.278644  0.004210
1                   0.042841            0.006863   -0.014630  0.003394
2                  -0.031806            0.005134    0.006089  0.020590
3                  -0.014942           -0.012421    0.009780 -0.023837

                    CreditCard         1         2         3
Age                   0.007681 -0.027770 -0.016264  0.045838
Experience            0.008830 -0.007549 -0.017334  0.025119
Income               -0.002385  0.218019 -0.128364 -0.108878
Family                0.011588 -0.118628  0.139201 -0.008744
CCAvg                -0.006689  0.156979 -0.090366 -0.080413
Mortgage             -0.007231  0.042841 -0.031806 -0.014942
Securities Account   -0.015028  0.006863  0.005134 -0.012421
CD Account            0.278644 -0.014630  0.006089  0.009780
Online                0.004210  0.003394  0.020590 -0.023837
CreditCard            1.000000  0.014925 -0.012196 -0.004113
1                     0.014925  1.000000 -0.530586 -0.556437
2                    -0.012196 -0.530586  1.000000 -0.409051
3                    -0.004113 -0.556437 -0.409051  1.000000
```

In [35]:

```python
# Specify the correlation threshold
correlation_threshold = 0.7

# List to store highly correlated variables
highly_correlated_variables = []

# Loop through each column in the correlation matrix
for col in correlation_matrix.columns:
    # Find variables highly correlated with the current variable
    correlated_vars = correlation_matrix.index[correlation_matrix[col].abs() > corr

    # Exclude the current variable itself
    correlated_vars.remove(col)

    # Append the variables to the list
    highly_correlated_variables.extend(correlated_vars)

# Remove duplicates from the list
highly_correlated_variables = list(set(highly_correlated_variables))
```

```
# Print the highly correlated variables
for variable in highly_correlated_variables:
    print(f"{variable} has a correlation coefficient above {correlation_threshold:.
```

```
Experience has a correlation coefficient above 0.70 with at least one variable.
Age has a correlation coefficient above 0.70 with at least one variable.
```

In [36]:

```
# Check if 'Experience' is in the columns
if 'Experience' in df.columns:
    # Drop the 'Experience' column if it exists
    df = df.drop('Experience', axis=1)
    print("'Experience' column has been dropped.")
else:
    print("'Experience' column not found in the dataset.")
```

```
'Experience' column has been dropped.
```

# Data Splitting

In [37]:

```
from sklearn.model_selection import train_test_split

# Assuming your features are stored in X and the target variable is in y
X = df.drop(columns=['Personal Loan'])
y = df['Personal Loan']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Print the shapes of the resulting sets
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (4000, 12)
X_test shape: (1000, 12)
y_train shape: (4000,)
y_test shape: (1000,)
```

In [38]:

```
X_train.head
```

Out[38]:
```
<bound method NDFrame.head of       Age     Income  Family  CCAvg  Mortgage  Secur
ities Account  CD Account  \
4227   32   9.250000       1    3.8         0                   1            0
4676   39   5.666667       3    2.1         0                   1            0
800    31  14.416667       1    6.0         0                   0            0
3671   50   1.500000       1    0.4         0                   0            0
4193   62   2.583333       3    0.2         0                   0            0
...   ...        ...     ...    ...       ...                 ...          ...
4426   33  11.666667       1    4.6         0                   0            0
466    25   1.083333       2    0.9         0                   0            0
3092   43   9.416667       2    0.4       325                   1            0
3772   35  12.666667       2    3.0         0                   0            0
860    57   2.500000       2    0.7       145                   0            0

      Online  CreditCard  1  2  3
4227       0           0  1  0  0
4676       1           0  1  0  0
800        1           0  1  0  0
3671       1           0  0  0  1
4193       1           0  1  0  0
...      ...         ... .. .. ..
4426       1           0  1  0  0
466        1           0  0  0  1
3092       0           0  1  0  0
3772       1           0  1  0  0
860        0           0  0  1  0

[4000 rows x 12 columns]>
```

In [39]:
```
y_train.head
```

Out[39]:
```
<bound method NDFrame.head of 4227    0
4676    0
800     0
3671    0
4193    0
       ..
4426    0
466     0
3092    0
3772    0
860     0
Name: Personal Loan, Length: 4000, dtype: int64>
```

In [40]:
```
print(y_train.value_counts())
```

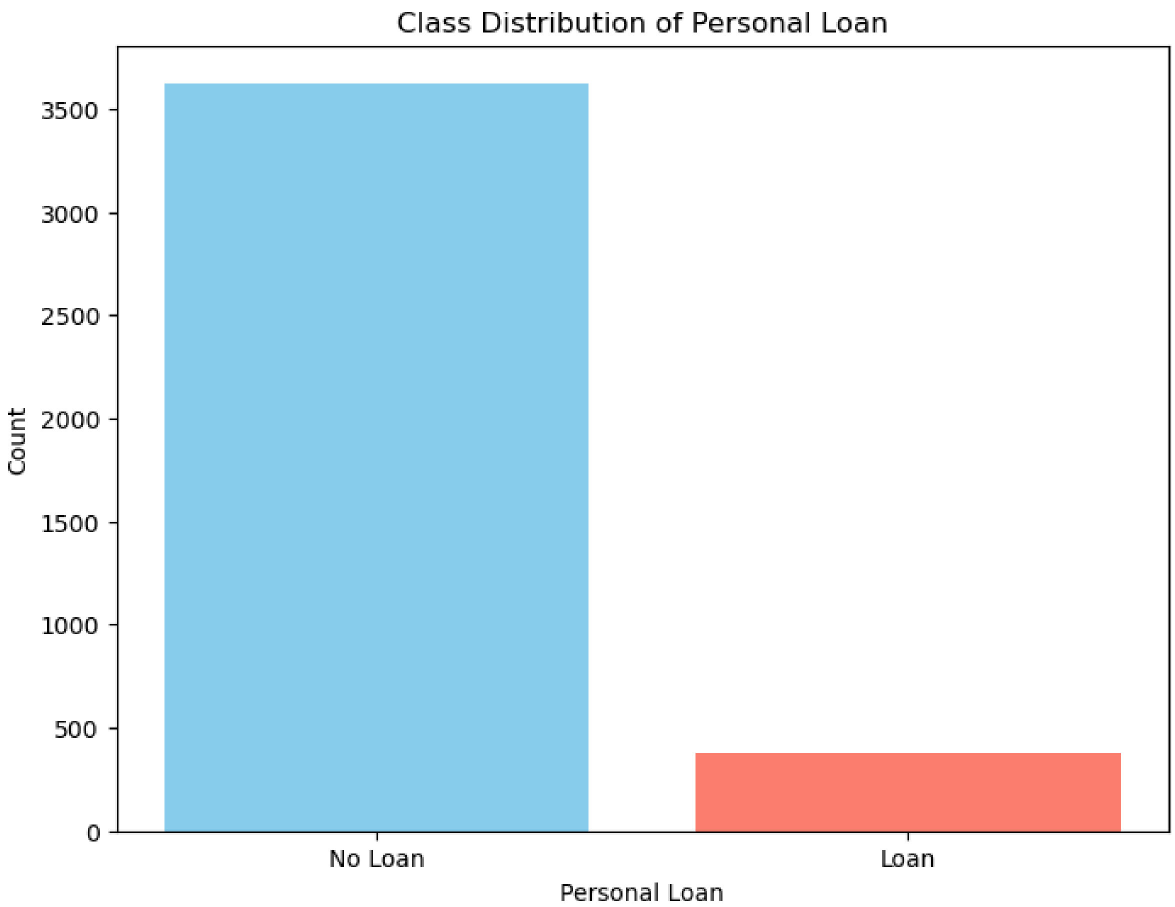
```
Personal Loan
0    3625
1     375
Name: count, dtype: int64
```

# Target Variable Distribution&Upsampling

In [41]:
```
#vcheck for imbalance with the Target Variable.
import pandas as pd
class_counts = y_train.value_counts()
# Display class distribution
print("Class Distribution:")
print(class_counts)
# Calculate imbalance ratio
imbalance_ratio = class_counts[0] / class_counts[1]
print("Imbalance Ratio:", imbalance_ratio)
```

```
Class Distribution:
Personal Loan
0    3625
1     375
Name: count, dtype: int64
Imbalance Ratio: 9.666666666666666
```

In [42]:
```
class_labels = ['No Loan', 'Loan']
class_counts = [sum(y_train == 0), sum(y_train == 1)]  # Count occurrences of each

plt.figure(figsize=(8, 6))
plt.bar(class_labels, class_counts, color=['skyblue', 'salmon'])
plt.title('Class Distribution of Personal Loan')
plt.xlabel('Personal Loan')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()
```




# Upsampling Using SMOTE

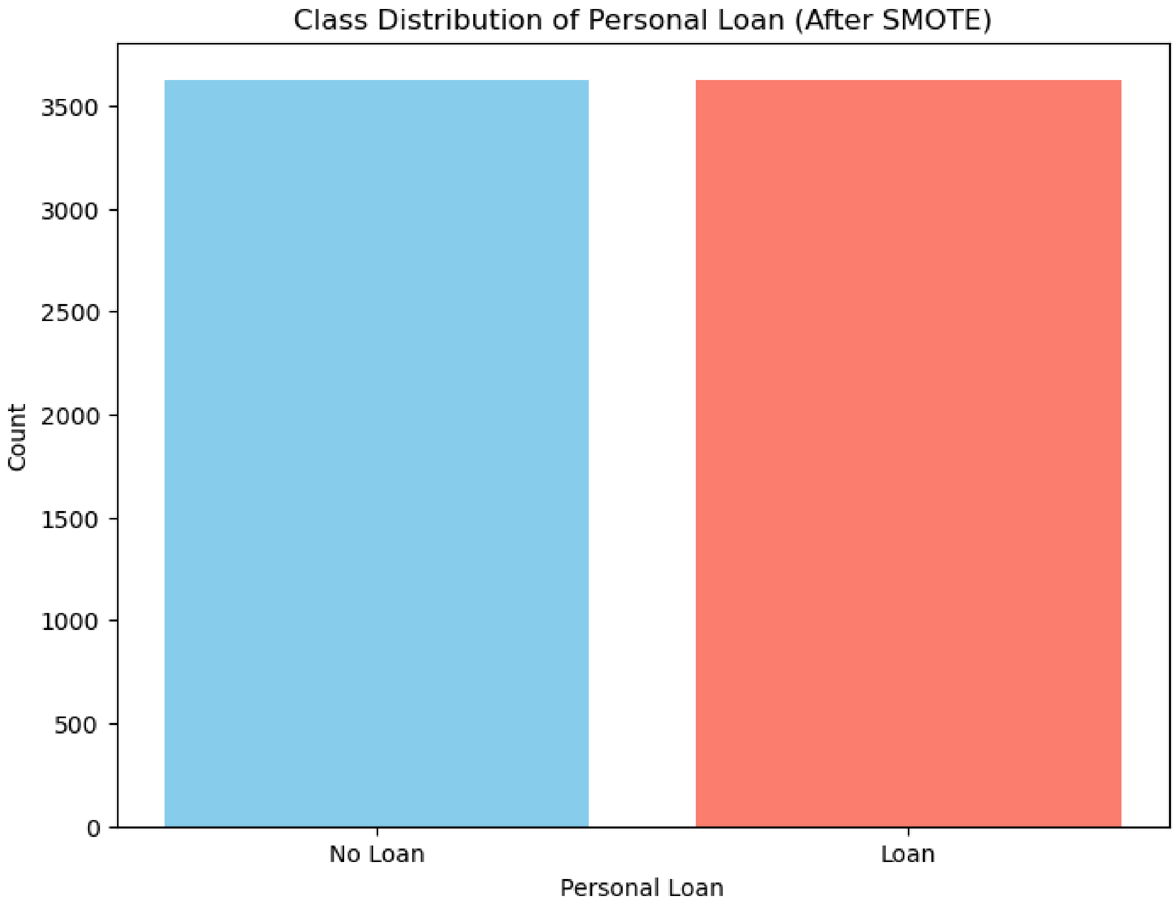
In [43]:
```
X_train.columns = X_train.columns.astype(str)
# Initialize SMOTE with a desired sampling strategy
smote = SMOTE(sampling_strategy='auto', random_state=42)
# Upsample the minority class in the training data
X_train, y_train = smote.fit_resample(X_train, y_train)
```

In [44]:
```
class_counts_SMOTE = y_train.value_counts()

print(class_counts_SMOTE)
```

```
Personal Loan
0    3625
1    3625
Name: count, dtype: int64
```

In [45]:

```
class_labels = ['No Loan', 'Loan']
class_counts = [sum(y_train == 0), sum(y_train == 1)]  # Count occurrences of each

plt.figure(figsize=(8, 6))
plt.bar(class_labels, class_counts, color=['skyblue', 'salmon'])
plt.title('Class Distribution of Personal Loan (After SMOTE)')
plt.xlabel('Personal Loan')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.savefig('Upsample.jpg')
plt.show()
```




In [46]:

```
X_train.dtypes
```

Out[46]:

```
Age                     int64
Income                float64
Family                  int64
CCAvg                 float64
Mortgage                int64
Securities Account      int64
CD Account              int64
Online                  int64
CreditCard              int64
1                       int32
2                       int32
3                       int32
dtype: object
```

In [47]:

```
# Convert NumPy arrays to Pandas DataFrames
X_train = pd.DataFrame(X_train)
```

```
X_test = pd.DataFrame(X_test)

# Convert column names to strings for both X_train and X_test
X_train.columns = X_train.columns.astype(str)
X_test.columns = X_test.columns.astype(str)
```

In [48]:
```
y_train_df = pd.DataFrame(y_train)
y_train_df.columns = y_train_df.columns.astype(str)
```

# Normalisation

In [49]:
```
# Normalize your oversampled train set
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [50]:
```
# Convert the scaled arrays back to DataFrames
X_train_scaled = pd.DataFrame(X_train)
X_test_scaled = pd.DataFrame(X_test)

# Display the first few rows of the scaled training data to see how the features we
print("Scaled Training Data:")
print(X_train_scaled.head())

# Display the first few rows of the scaled testing data to see how the features wer
print("\nScaled Testing Data:")
print(X_test_scaled.head())
```

```
Scaled Training Data:
          0         1         2     3    4    5    6    7    8    9   10   11
0  0.204545  0.476852  0.000000  0.38  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0
1  0.363636  0.277778  0.666667  0.21  0.0  1.0  0.0  1.0  0.0  1.0  0.0  0.0
2  0.181818  0.763889  0.000000  0.60  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0
3  0.613636  0.046296  0.000000  0.04  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0
4  0.886364  0.106481  0.666667  0.02  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0

Scaled Testing Data:
          0         1         2     3         4    5    6    7    8    9   10  \
0  0.159091  0.125000  0.333333  0.03  0.000000  1.0  0.0  0.0  1.0  0.0  1.0
1  0.545455  0.652778  1.000000  0.61  0.000000  0.0  0.0  0.0  1.0  1.0  0.0
2  0.159091  0.523148  0.333333  0.31  0.661264  0.0  0.0  1.0  0.0  1.0  0.0
3  0.181818  0.250000  0.000000  0.10  0.000000  1.0  0.0  1.0  0.0  1.0  0.0
4  0.886364  0.101852  0.666667  0.07  0.000000  0.0  0.0  1.0  0.0  0.0  1.0

    11
0  0.0
1  0.0
2  0.0
3  0.0
4  0.0
```

# Machine Learning Algorithms

## Decision Tree Classifier

In [51]:
```
#Decision Tree Classifier Using Gini Index and Entropy
#using Gini Index
```

```
# Creating the Decision Tree classifer object with splitting criterion as Gini inde
DTC = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)
```

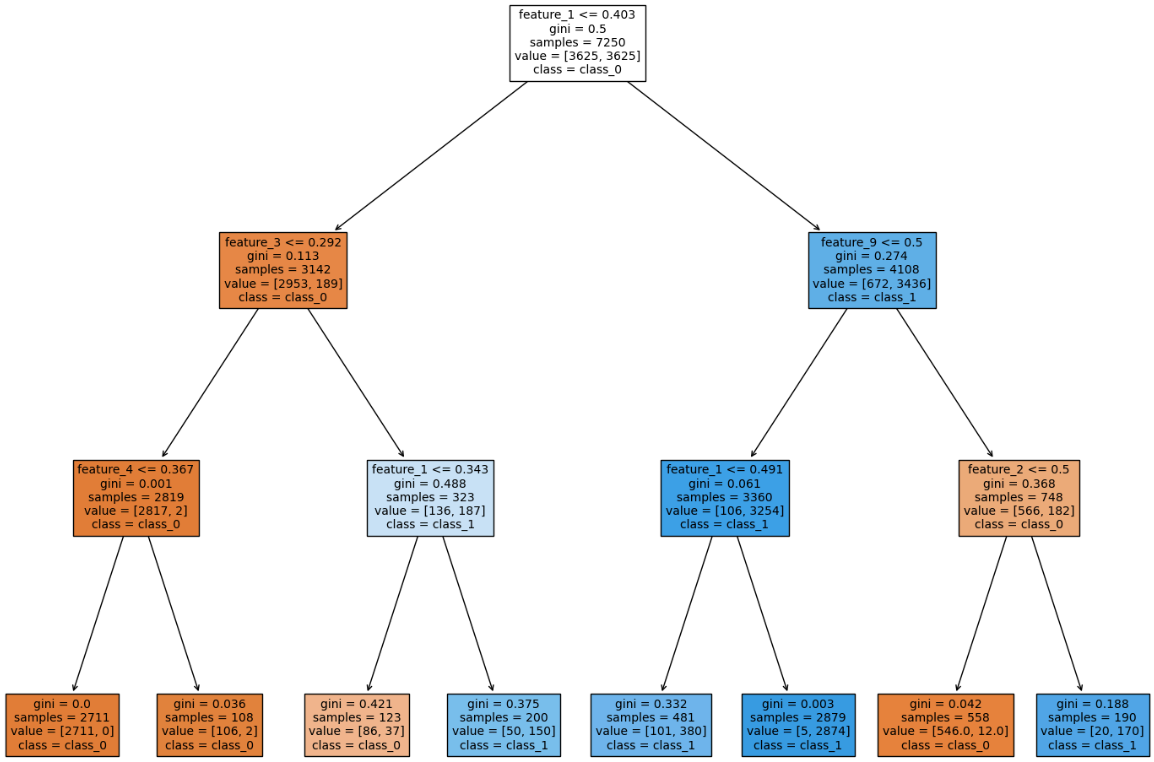
In [52]:
```
# Training Decision Tree Classifer
DTC = DTC.fit(X_train, y_train)
Y_predict_gini_test = DTC.predict(X_test)
```

In [53]:
```
# Calculating accuracy
accuracy_gini = accuracy_score(y_test, Y_predict_gini_test)
print("Accuracy:", accuracy_gini)
```

```
Accuracy: 0.957
```

In [54]:
```
#Apply grid search here
```

In [55]:
```
# Creating the Decision Tree classifier object with splitting criterion as Gini ind
DTC = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)

DTC.fit(X_train, y_train)

# Get feature names based on the dimensions of X_train
n_features = X_train.shape[1]
feature_names = [f"feature_{i}" for i in range(n_features)]

# Plotting Gini Tree with larger dimensions and adjusted font size
plt.figure(figsize=(18, 14))  # Adjust dimensions as needed
plot_tree(DTC, filled=True, feature_names=feature_names, class_names=['class_0', 'c
plt.savefig('Decision_tree_gini_plot.jpg')
plt.show()
```




In [56]:
```
#Predicting the accuracy on train set for comparision
Y_predict_gini_train = DTC.predict(X_train)
```

In [57]:
```
Y_predict_gini_train
```

Out[57]:
```
array([0, 0, 0, ..., 1, 1, 1], dtype=int64)
```

In [58]:
```
print("Accuracy with gini index:",metrics.accuracy_score(y_test, Y_predict_gini_tes
print("Accuracy on training data (using Gini index criterion):", metrics.accuracy_s
```

```
Accuracy with gini index: 0.957
Accuracy on training data (using Gini index criterion): 0.9686896551724138
```

In [59]:
```
# Print classification report
print("Decision Tree Classification Report (Using Gini Index) on Test Data:")
print(classification_report(y_test, Y_predict_gini_test))
print("Decision Tree Classification Report (Using Gini Index) on Train Data:")
print(classification_report(y_train, Y_predict_gini_train))
```

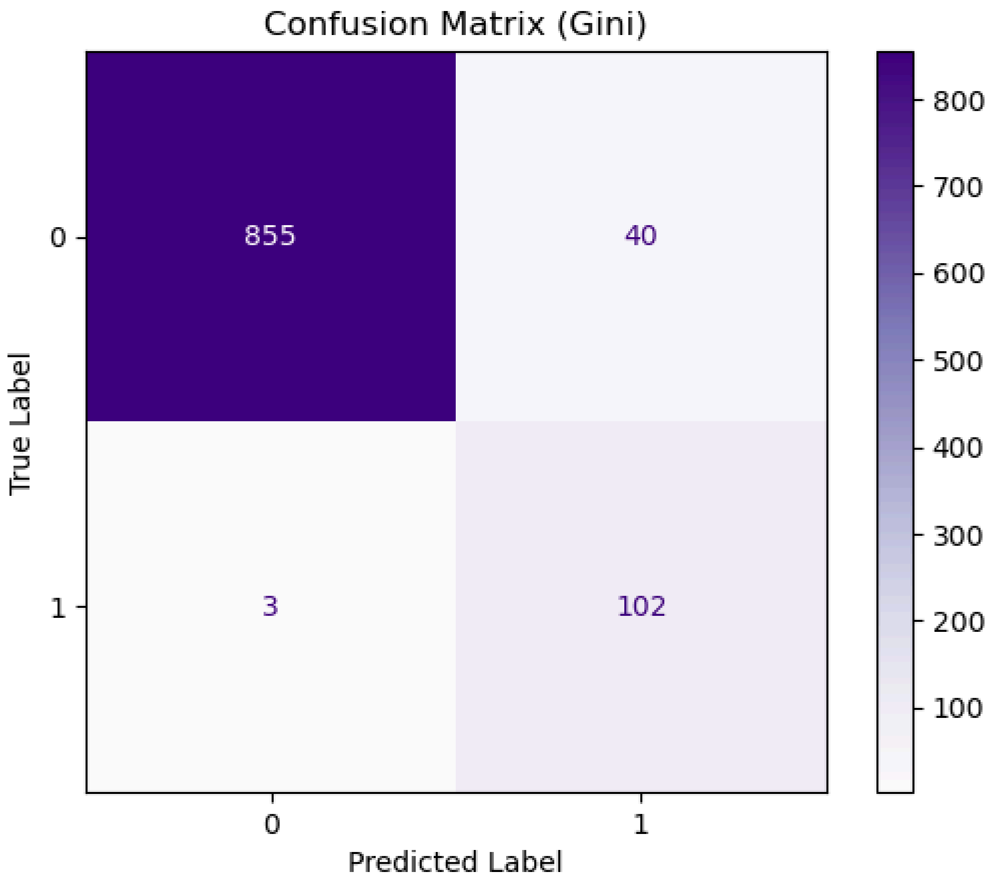
```
Decision Tree Classification Report (Using Gini Index) on Test Data:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       895
           1       0.72      0.97      0.83       105

    accuracy                           0.96      1000
   macro avg       0.86      0.96      0.90      1000
weighted avg       0.97      0.96      0.96      1000

Decision Tree Classification Report (Using Gini Index) on Train Data:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97      3625
           1       0.95      0.99      0.97      3625

    accuracy                           0.97      7250
   macro avg       0.97      0.97      0.97      7250
weighted avg       0.97      0.97      0.97      7250
```

In [60]:
```
# Train a Decision Tree classifier with Gini criterion
dt_gini_model = DecisionTreeClassifier(criterion='gini')
dt_gini_model.fit(X_train, y_train)

# Predicting on the test set using the trained Decision Tree (Gini) model
y_pred_dt_gini = dt_gini_model.predict(X_test)

# Calculating accuracy
accuracy_dt_gini = accuracy_score(y_test, y_pred_dt_gini)
print("Accuracy_dt_gini:", accuracy_dt_gini)

# Calculating precision
precision_dt_gini = precision_score(y_test, y_pred_dt_gini)
print("Precision_dt_gini:", precision_dt_gini)

# Calculating recall
recall_dt_gini = recall_score(y_test, y_pred_dt_gini)
print("Recall_dt_gini:", recall_dt_gini)

# Calculating F1 score
f1_dt_gini = f1_score(y_test, y_pred_dt_gini)
print("F1 Score_dt_gini:", f1_dt_gini)
```

```
Accuracy_dt_gini: 0.984
Precision_dt_gini: 0.8869565217391304
Recall_dt_gini: 0.9714285714285714
F1 Score_dt_gini: 0.9272727272727272
```

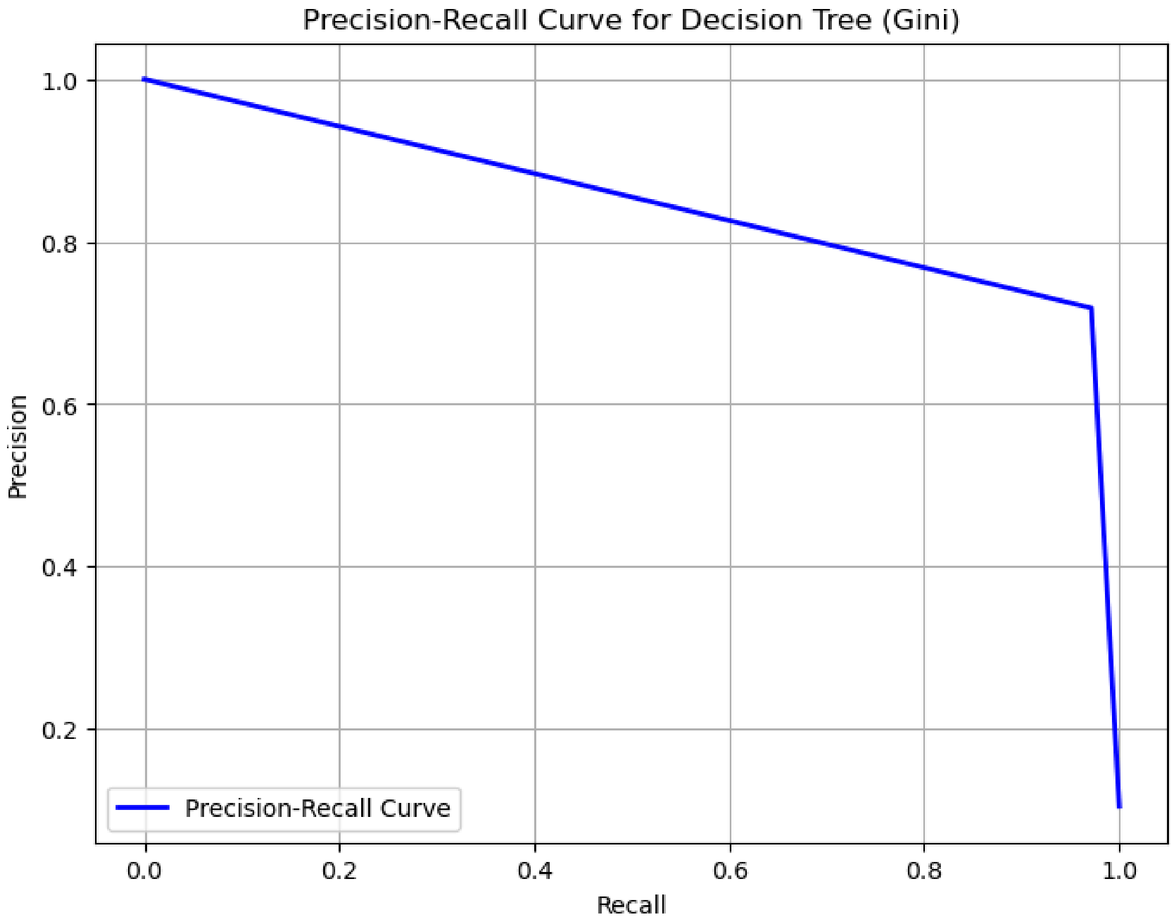
# Confusion Matrix(Gini)

In [61]:
```
# Display the confusion matrix for the model
cm_gini = metrics.confusion_matrix(y_test, Y_predict_gini_test)
disp_gini = metrics.ConfusionMatrixDisplay(confusion_matrix=cm_gini)
disp_gini.plot(cmap='Purples')
plt.title('Confusion Matrix (Gini)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



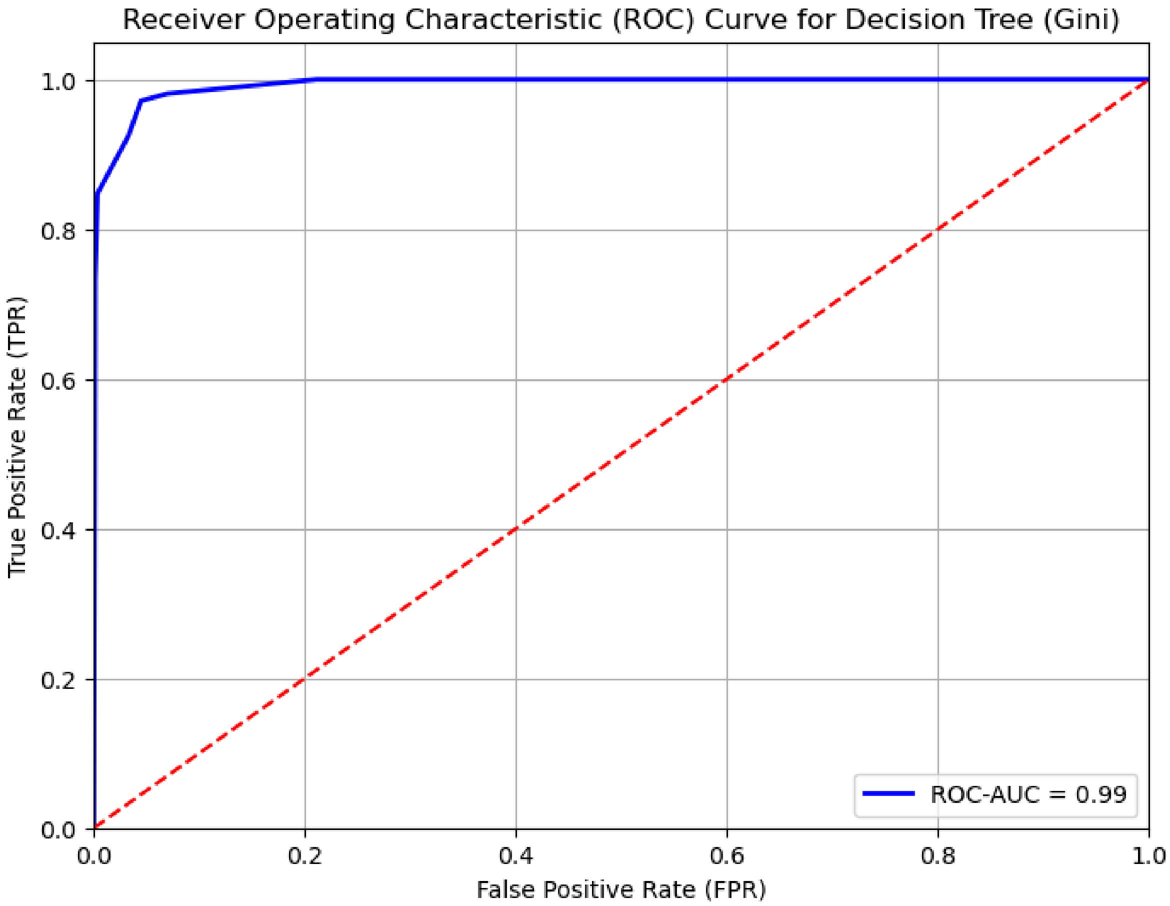

# Precision-Recall(Gini)

In [62]:
```
# Get precision, recall, and thresholds
precision_gini, recall_gini, thresholds_gini = precision_recall_curve(y_test, Y_pre

# Plot precision-recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall_gini, precision_gini, color='blue', lw=2, label='Precision-Recall (
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Decision Tree (Gini)')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()
```




# Roc Curve(Gini)

In [63]:
```
# Get predicted probabilities for the positive class
Y_pred_proba_gini = DTC.predict_proba(X_test)[:, 1]

# Calculate fpr, tpr, and thresholds
fpr_gini, tpr_gini, thresholds_gini = roc_curve(y_test, Y_pred_proba_gini)

# Calculate ROC-AUC score
roc_auc_gini = roc_auc_score(y_test, Y_pred_proba_gini)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_gini, tpr_gini, color='blue', lw=2, label=f'ROC-AUC = {roc_auc_gini:.2
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line representing
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve for Decision Tree (Gini)')
plt.legend(loc='lower right')  # Display ROC-AUC score in the legend
plt.grid(True)
plt.show()
```




In [64]:

```
#Decision Tree classifier using Entropy
# Creating the Decision Tree classifier object with splitting criterion as Entropy,
DTC_Entropy = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state

# Training Decision Tree Classifier
DTC_Entropy = DTC_Entropy.fit(X_train, y_train)
```

In [65]:

```
#Predicting the accuracy on test set for comparision
Y_predict_Entropy = DTC_Entropy.predict(X_test)
```

In [66]:

```
# Print classification report
print("Decision Tree Classification Report (Using Entropy) on Test Data:")
print(classification_report(y_test, Y_predict_Entropy))
```

```
Decision Tree Classification Report (Using Entropy) on Test Data:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97       895
           1       0.65      0.97      0.78       105

    accuracy                           0.94      1000
   macro avg       0.82      0.95      0.87      1000
weighted avg       0.96      0.94      0.95      1000
```

In [67]:

```
# Assuming you have already defined and split your data into X_train, X_test, y_tra

# Train a Decision Tree classifier with Entropy criterion
dt_entropy_model = DecisionTreeClassifier(criterion='entropy')
dt_entropy_model.fit(X_train, y_train)

# Predicting on the test set using the trained Decision Tree (Entropy) model
y_pred_dt_entropy = dt_entropy_model.predict(X_test)
```

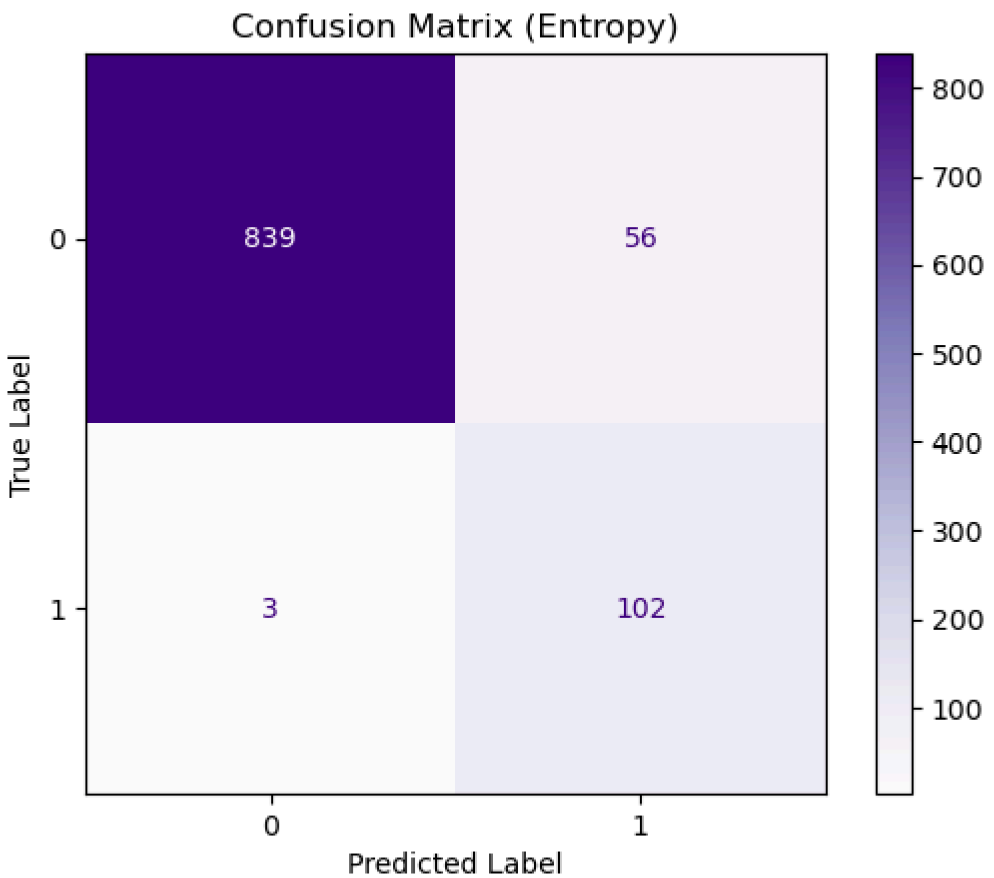
```
# Calculating accuracy
accuracy_dt_entropy = accuracy_score(y_test, y_pred_dt_entropy)
print("Accuracy_dt_entropy:", accuracy_dt_entropy)

# Calculating precision
precision_dt_entropy = precision_score(y_test, y_pred_dt_entropy)
print("Precision_dt_entropy:", precision_dt_entropy)

# Calculating recall
recall_dt_entropy = recall_score(y_test, y_pred_dt_entropy)
print("Recall_dt_entropy:", recall_dt_entropy)

# Calculating F1 score
f1_dt_entropy = f1_score(y_test, y_pred_dt_entropy)
print("F1 Score_dt_entropy:", f1_dt_entropy)
```

```
Accuracy_dt_entropy: 0.99
Precision_dt_entropy: 0.9357798165137615
Recall_dt_entropy: 0.9714285714285714
F1 Score_dt_entropy: 0.9532710280373832
```

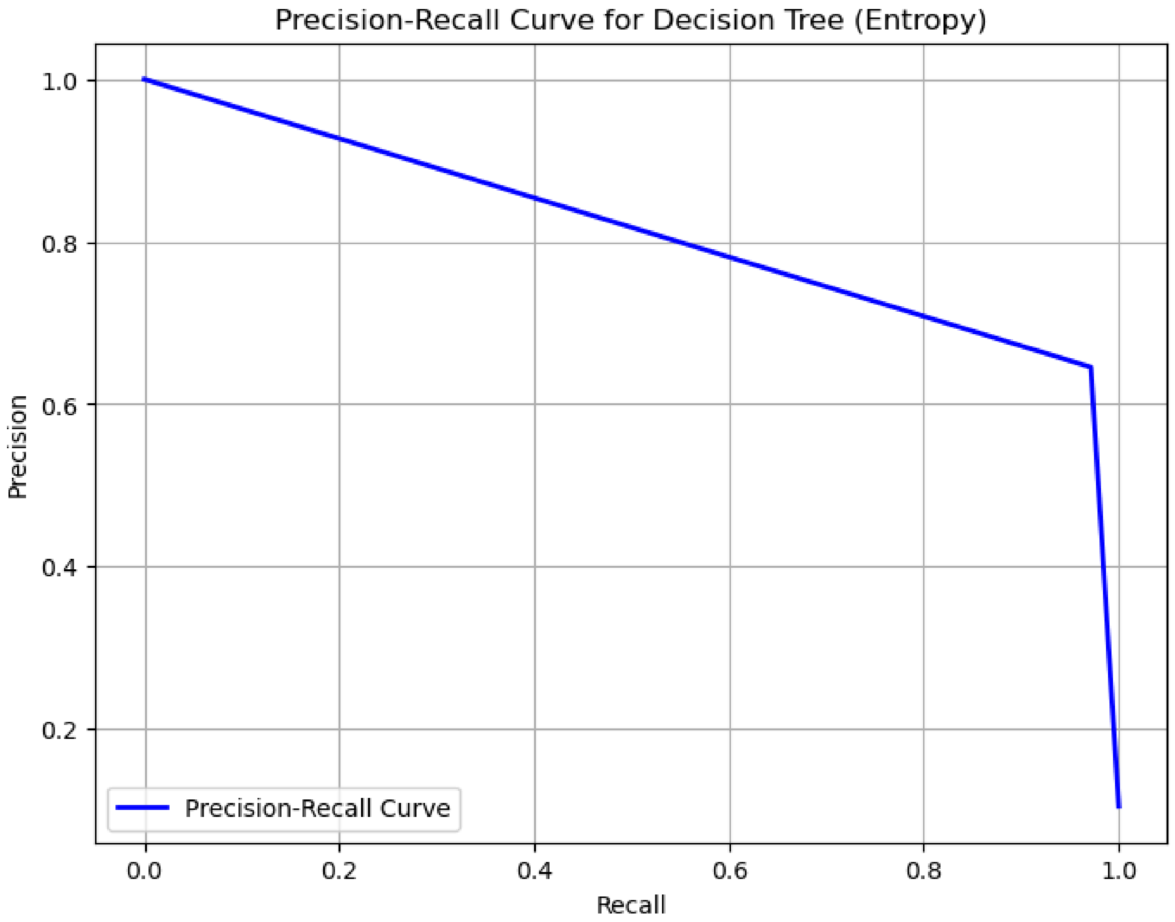
# Confusion Matrix for Decision Tree(Entropy)

In [68]:

```
# Display the confusion matrix for the model
cm_Entropy = metrics.confusion_matrix(y_test, Y_predict_Entropy)
disp_Entropy = metrics.ConfusionMatrixDisplay(confusion_matrix=cm_Entropy)
disp_Entropy.plot(cmap='Purples')
plt.title('Confusion Matrix (Entropy)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



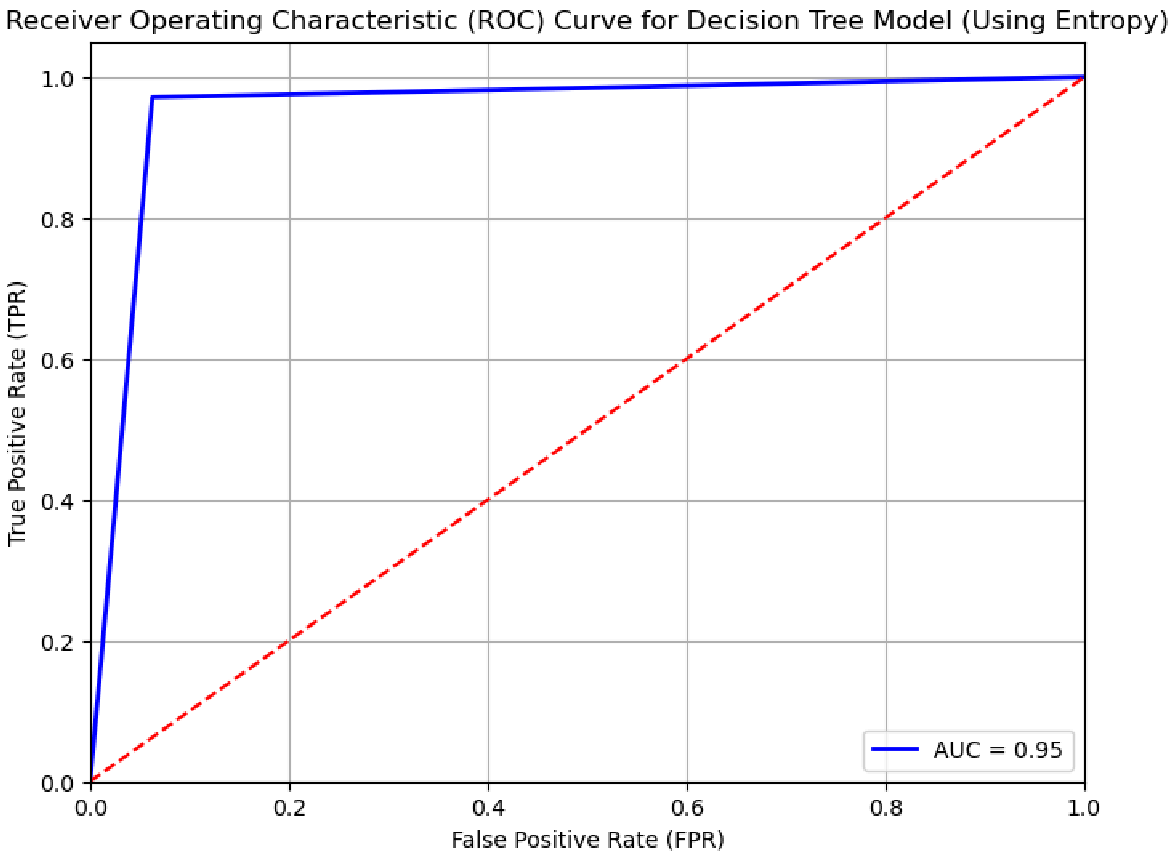
In [69]:
```
# Get precision, recall, and thresholds
precision_entropy, recall_entropy, thresholds_entropy = precision_recall_curve(y_te

# Plot precision-recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall_entropy, precision_entropy, color='blue', lw=2, label='Precision-Re
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Decision Tree (Entropy)')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()
```




# Roc Curve For Decision Tree(Entropy)

In [70]:
```
fpr, tpr, thresholds = roc_curve(y_test, Y_predict_Entropy)

# Calculate the AUC score
auc_score = roc_auc_score(y_test, Y_predict_Entropy)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {auc_score:.2f}')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line representing
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve for Decision Tree Model (U
plt.legend(loc='lower right')  # Display AUC score in the legend
plt.grid(True)
```

```
plt.savefig('Receiver_Operating_Characteristic_ROC_Curve_Decision_Tree_Model_Using_
plt.show()
```




# Logistic Regression

In [71]:
```
# Train a Logistic Regression Model (using the default parameters)
log_model = LogisticRegression(random_state=16, max_iter=1000)

# Fit the model with data
log_model.fit(X_train, y_train)
```

Out[71]:
```
LogisticRegression
LogisticRegression(max_iter=1000, random_state=16)
```

In [72]:
```
y_pred_log = log_model.predict(X_test)
```

In [73]:
```
#printing accuracy
accuracy_log = accuracy_score(y_test, y_pred_log)
precision_log = precision_score(y_test, y_pred_log)
recall_log = recall_score(y_test, y_pred_log)
f1_log = f1_score(y_test, y_pred_log)
print("Accuracy:", accuracy_log)
print("Precision:", precision_log)
print("Recall:", recall_log)
print("F1 Score:", f1_log)
# Print classification report
print("Logistic Regression Classification Report before optimisation:")
print(classification_report(y_test, y_pred_log))
plt.show()
```

```
Accuracy: 0.944
Precision: 0.6870229007633588
Recall: 0.8571428571428571
F1 Score: 0.7627118644067796
Logistic Regression Classification Report before optimisation:
              precision    recall  f1-score   support

           0       0.98      0.95      0.97       895
           1       0.69      0.86      0.76       105

    accuracy                           0.94      1000
   macro avg       0.83      0.91      0.87      1000
weighted avg       0.95      0.94      0.95      1000
```

In [74]:

```
# Convert NumPy arrays to Pandas DataFrames
X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)

# Convert column names to strings for both X_train and X_test
X_train.columns = X_train.columns.astype(str)
X_test.columns = X_test.columns.astype(str)

# Initialize the Logistic Regression model
logreg_model = LogisticRegression(random_state=42)

# Train the model on the training data
logreg_model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = logreg_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
plt.show()
```

```
Accuracy: 0.944
Confusion Matrix:
 [[854  41]
 [ 15  90]]
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.95      0.97       895
           1       0.69      0.86      0.76       105

    accuracy                           0.94      1000
   macro avg       0.83      0.91      0.87      1000
weighted avg       0.95      0.94      0.95      1000
```

# Optimization Of Logistic Regression

In [75]:

```
# Define the parameter grid
param_grid = {
```

```
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization strength
    'penalty': ['l1', 'l2'],               # Penalty type
    'solver': ['liblinear', 'saga']        # Solver algorithm
}
```

In [76]:
```
warnings.filterwarnings("ignore", category=ConvergenceWarning)
# Creating a logistic regression model
logistic_regression = LogisticRegression(max_iter=1000, random_state=16)
# The GridSearchCV object
grid_search = GridSearchCV(logistic_regression, param_grid, cv=5, scoring='accuracy
# Performing grid search to find the best hyperparameters
grid_search.fit(X_train, y_train)
```

Out[76]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [77]:
```
# Printing the best hyperparameters found
print("Best hyperparameters:", grid_search.best_params_)
```

```
Best hyperparameters: {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
```

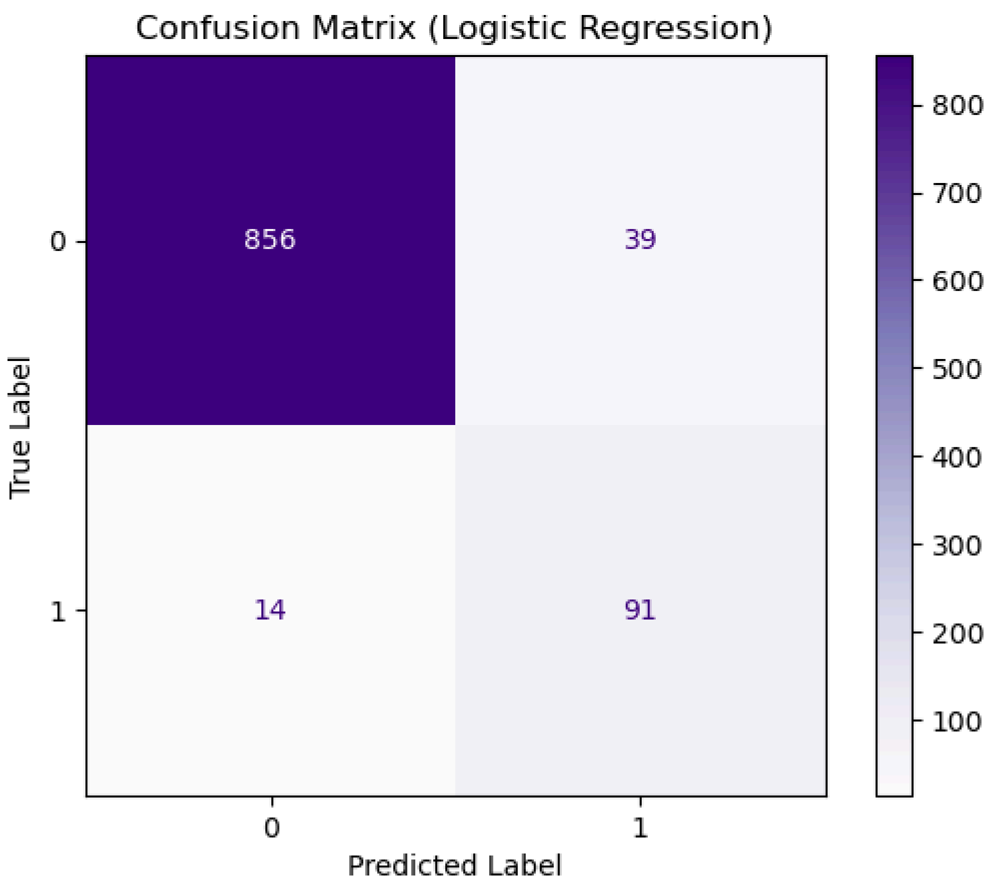
In [78]:
```
# Get the best model
best_model = grid_search.best_estimator_
# Evaluate the best model on the test set
y_pred_best = best_model.predict(X_test)
```

In [79]:
```
# Calculate accuracy
accuracy_logreg = accuracy_score(y_test, y_pred_best)
precision_logreg = precision_score(y_test, y_pred_best)
recall_logreg = recall_score(y_test, y_pred_best)
f1_logreg = f1_score(y_test, y_pred_best)  # Using y_test instead of Y_test
print("Accuracy of best model:", accuracy_logreg)
print("Precision of best model:", precision_logreg)
print("Recall of best model:", recall_logreg)
print("F1 of best model:", f1_logreg)

# Print classification report
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_best))
```

```
Accuracy of best model: 0.947
Precision of best model: 0.7
Recall of best model: 0.8666666666666667
F1 of best model: 0.774468085106383
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       895
           1       0.70      0.87      0.77       105

    accuracy                           0.95      1000
   macro avg       0.84      0.91      0.87      1000
weighted avg       0.95      0.95      0.95      1000
```

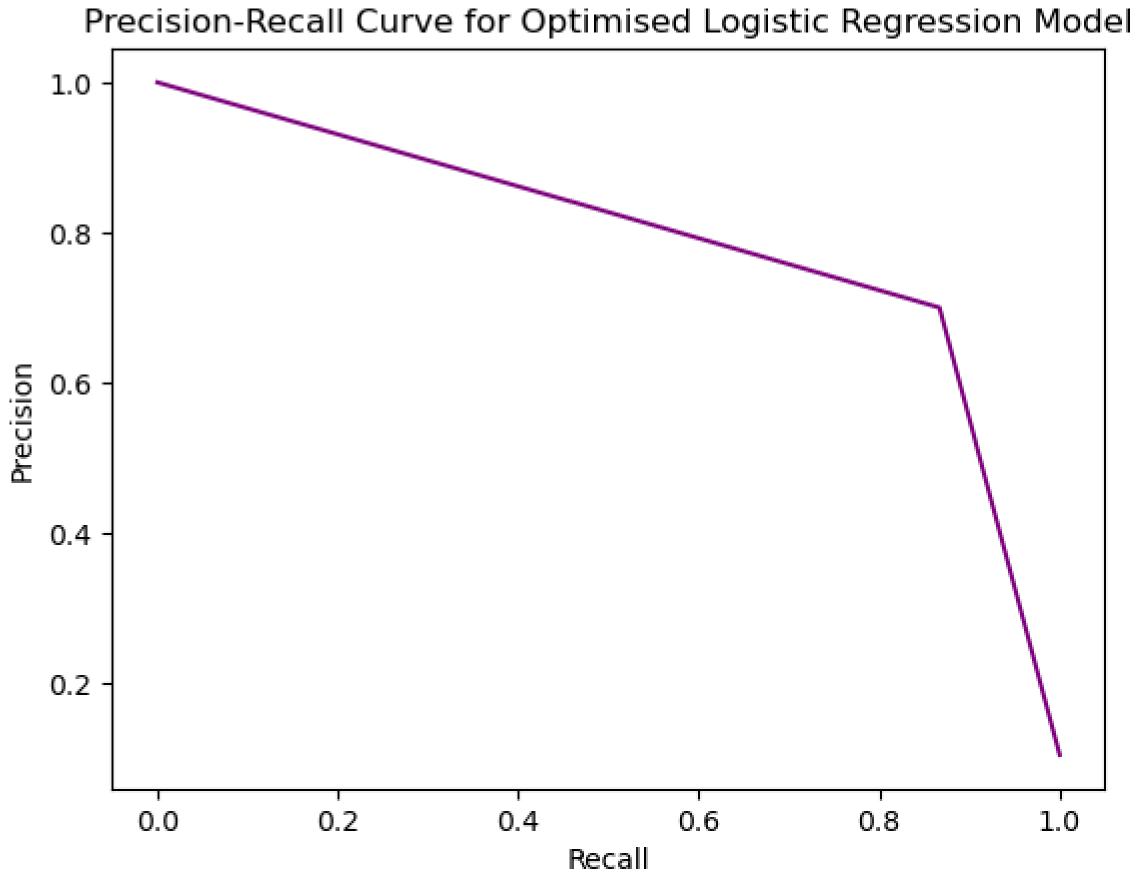
In [80]:
```
# Plot the confusion matrix for the Logistic Regression Model
cm_log = confusion_matrix(y_test, y_pred_best)
disp_log = metrics.ConfusionMatrixDisplay(confusion_matrix=cm_log)
```

```
disp_log.plot(cmap='Purples')
plt.title('Confusion Matrix (Logistic Regression)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('Confusion Matrix for Logistic Regression.jpg')
plt.show()
```

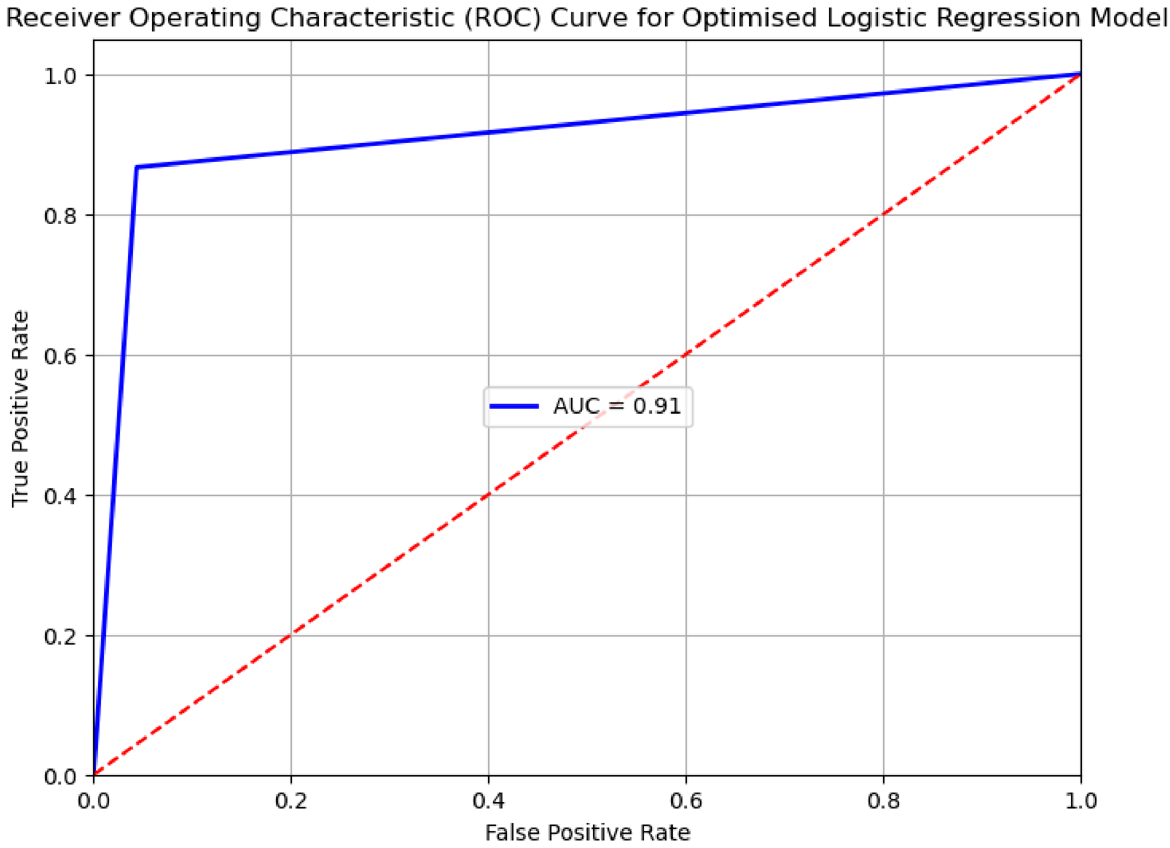



In [81]:

```
# Calculate Precision and Recall for the optimized model
precision_best, recall_best, thresholds_best = precision_recall_curve(y_test, y_pre
# Create Precision-Recall Curve
fig, ax = plt.subplots()
ax.plot(recall_best, precision_best, color='purple')
# Adding axis labels and title to the plot
ax.set_title('Precision-Recall Curve for Optimised Logistic Regression Model')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
# Save the plot as an image file and display
plt.savefig('Precision-Recall Curve for Optimised Logistic Regression Model.jpg')
plt.show()
```




In [82]:
```
# Calculate False Positive Rate, True Positive Rate, and Thresholds for the optimiz
fpr_best, tpr_best, thresholds_best = roc_curve(y_test, y_pred_best)
# Calculate the AUC score for the optimized model
auc_score_best = roc_auc_score(y_test, y_pred_best)
# Plot ROC curve for the optimized model
plt.figure(figsize=(8, 6))
plt.plot(fpr_best, tpr_best, color='blue', lw=2, label=f'AUC = {auc_score_best:.2f}
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Optimised Logistic Reg
plt.legend(loc='center')
plt.grid(True)
plt.savefig('Receiver Operating Characteristic (ROC) Curve for Optimised Logistic R
plt.show()
```




# Naive Bayes

In [83]:
```
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
```

In [84]:
```
nb_model.fit(X_train, y_train)
```

Out[84]:
```
GaussianNB
GaussianNB()
```

In [85]:
```
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matr
```

In [86]:
```
y_pred_nb = nb_model.predict(X_test)
```

In [87]:
```
print("Classification Report:")
print(classification_report(y_test, y_pred_nb))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       895
           1       0.66      0.77      0.71       105

    accuracy                           0.93      1000
   macro avg       0.82      0.86      0.84      1000
weighted avg       0.94      0.93      0.94      1000
```

In [88]:
```
# Train a Naive Bayes classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

```
# Predicting on the test set using the trained Naive Bayes model
y_pred = nb_model.predict(X_test)

# Calculating accuracy
accuracy_nb = accuracy_score(y_test, y_pred)
print("Accuracy_nb:", accuracy_nb)

# Calculating precision
precision_nb = precision_score(y_test, y_pred)
print("Precision_nb:", precision_nb)

# Calculating recall
recall_nb = recall_score(y_test, y_pred)
print("Recall_nb:", recall_nb)

# Calculating F1 score
f1_nb = f1_score(y_test, y_pred)
print("F1 Score_nb:", f1_nb)
```

```
Accuracy_nb: 0.934
Precision_nb: 0.6585365853658537
Recall_nb: 0.7714285714285715
F1 Score_nb: 0.7105263157894737
```

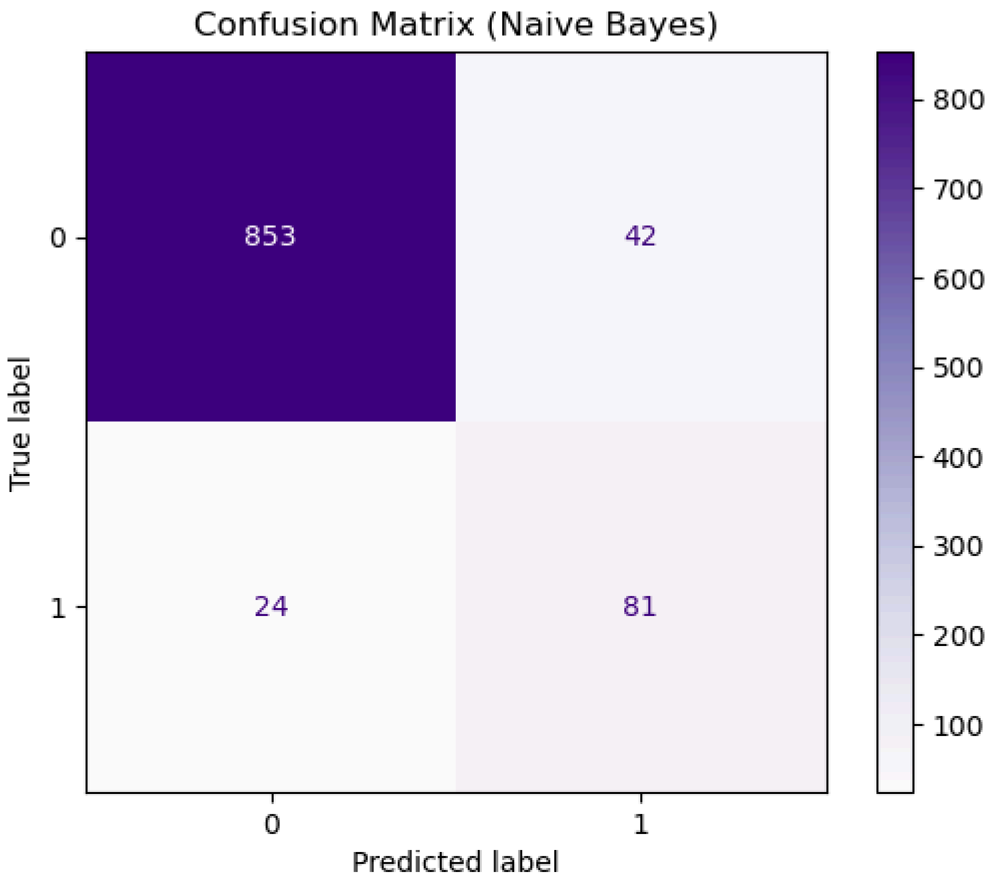
In [89]:

```
conf_mat_nb = confusion_matrix(y_test, y_pred_nb)
print("Confusion Matrix:")
print(conf_mat_nb)
```

```
Confusion Matrix:
[[853  42]
 [ 24  81]]
```

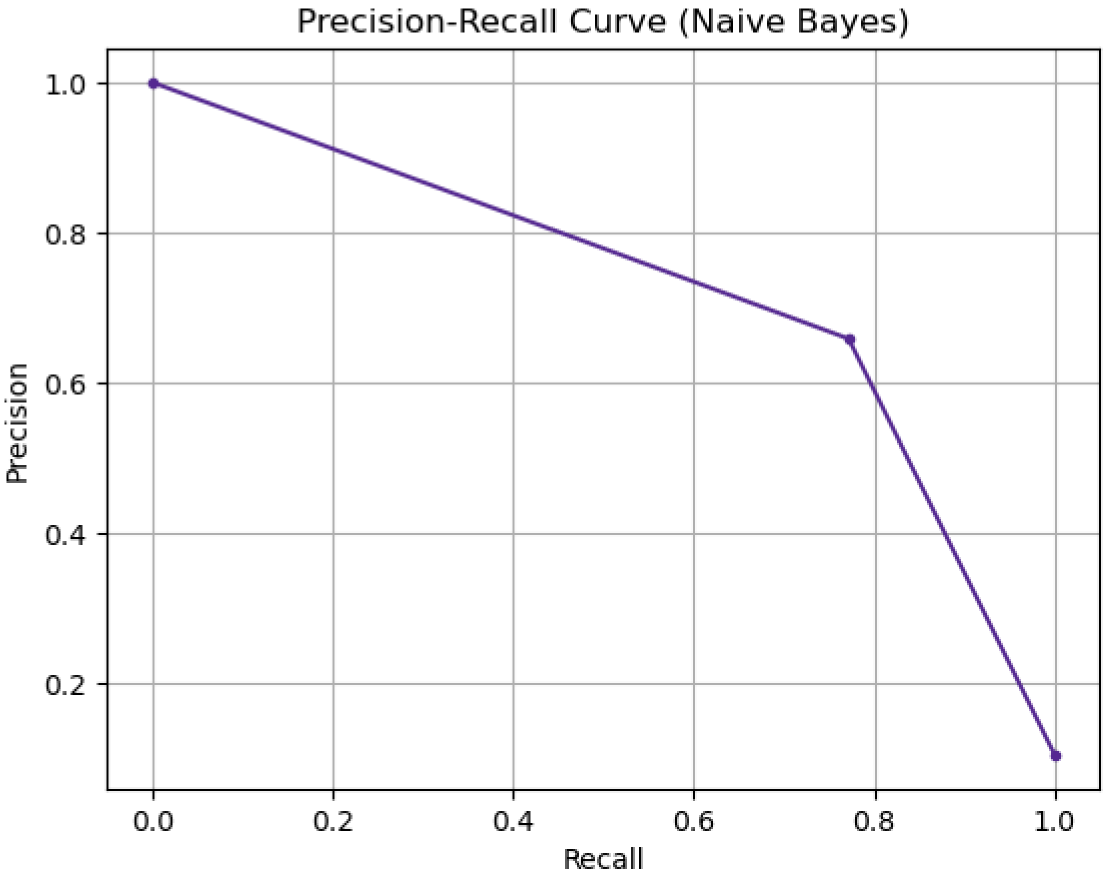
In [90]:

```
cm_nb = confusion_matrix(y_test, y_pred_nb)
# Display the confusion matrix
disp_nb = metrics.ConfusionMatrixDisplay(confusion_matrix=cm_nb)
disp_nb.plot(cmap='Purples')
# Customize the plot with true positives, false positives, true negatives, and fals
plt.title('Confusion Matrix (Naive Bayes)')
plt.show()
```

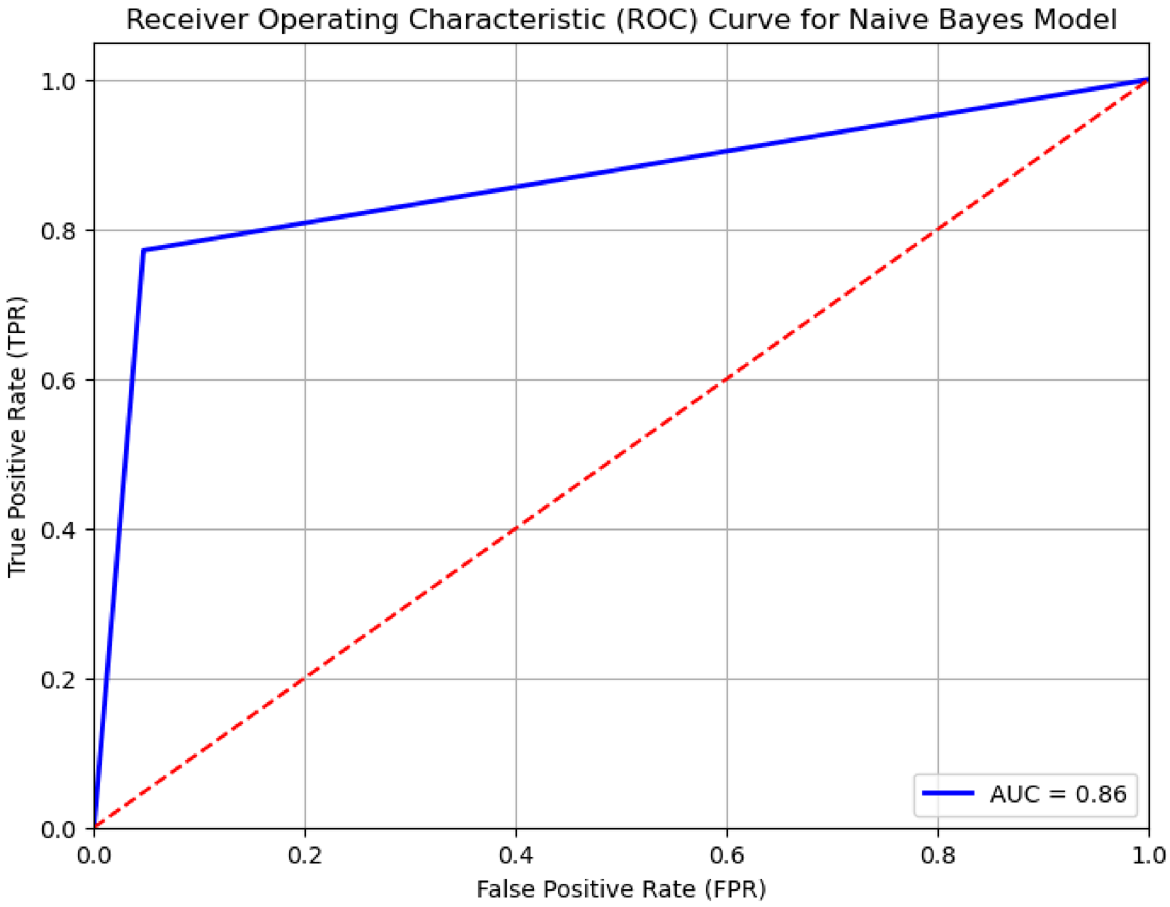



In [91]:

```
# Calculate precision and recall values
precision_nb, recall_nb, _ = precision_recall_curve(y_test, y_pred_nb)

# Plot precision-recall curve
plt.plot(recall_nb, precision_nb, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (Naive Bayes)')
plt.grid(True)
plt.show()
```




In [92]:
```
fpr_nb, tpr_nb, thresholds_nb = roc_curve(y_test, y_pred_nb)
auc_score_nb = roc_auc_score(y_test, y_pred_nb)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_nb, tpr_nb, color='blue', lw=2, label=f'AUC = {auc_score_nb:.2f}')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line representing
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve for Naive Bayes Model')
plt.legend(loc='lower right')  # Display AUC score in the legend
plt.grid(True)
plt.show()
```




# Random Forest Classifier

In [93]:
```
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

Out[93]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [94]:
```
y_pred = model.predict(X_test)
```

In [95]:
```
precision = precision_score(y_test, y_pred, average='binary')
recall = recall_score(y_test, y_pred, average='binary')
f1 = f1_score(y_test, y_pred, average='binary')
```

In [96]:
```
# Assuming you have already defined and split your data into X_train, X_test, y_tra

# Train a Random Forest classifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Predicting on the test set using the trained Random Forest model
y_pred_rf = rf_model.predict(X_test)

# Calculating accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy_rf:", accuracy_rf)

# Calculating precision
precision_rf = precision_score(y_test, y_pred_rf)
```

```
print("Precision_rf:", precision_rf)

# Calculating recall
recall_rf = recall_score(y_test, y_pred_rf)
print("Recall_rf:", recall_rf)

# Calculating F1 score
f1_rf = f1_score(y_test, y_pred_rf)
print("F1 Score_rf:", f1_rf)
```

```
Accuracy_rf: 0.991
Precision_rf: 0.9615384615384616
Recall_rf: 0.9523809523809523
F1 Score_rf: 0.9569377990430622
```

In [97]:

```
conf_mat = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_mat)
```

```
Confusion Matrix:
[[892   3]
 [  5 100]]
```

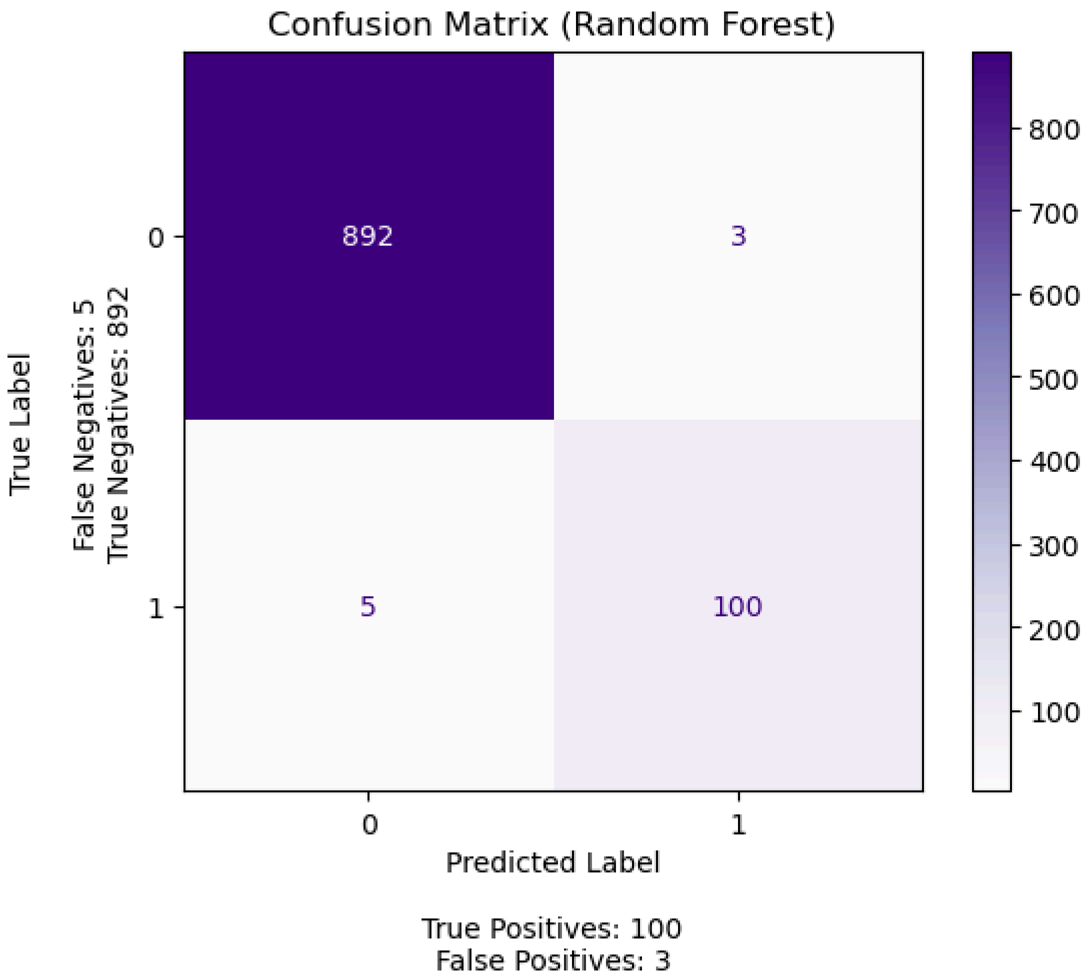
In [98]:

```
print('Classification Report:')
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       895
           1       0.97      0.95      0.96       105

    accuracy                           0.99      1000
   macro avg       0.98      0.97      0.98      1000
weighted avg       0.99      0.99      0.99      1000
```

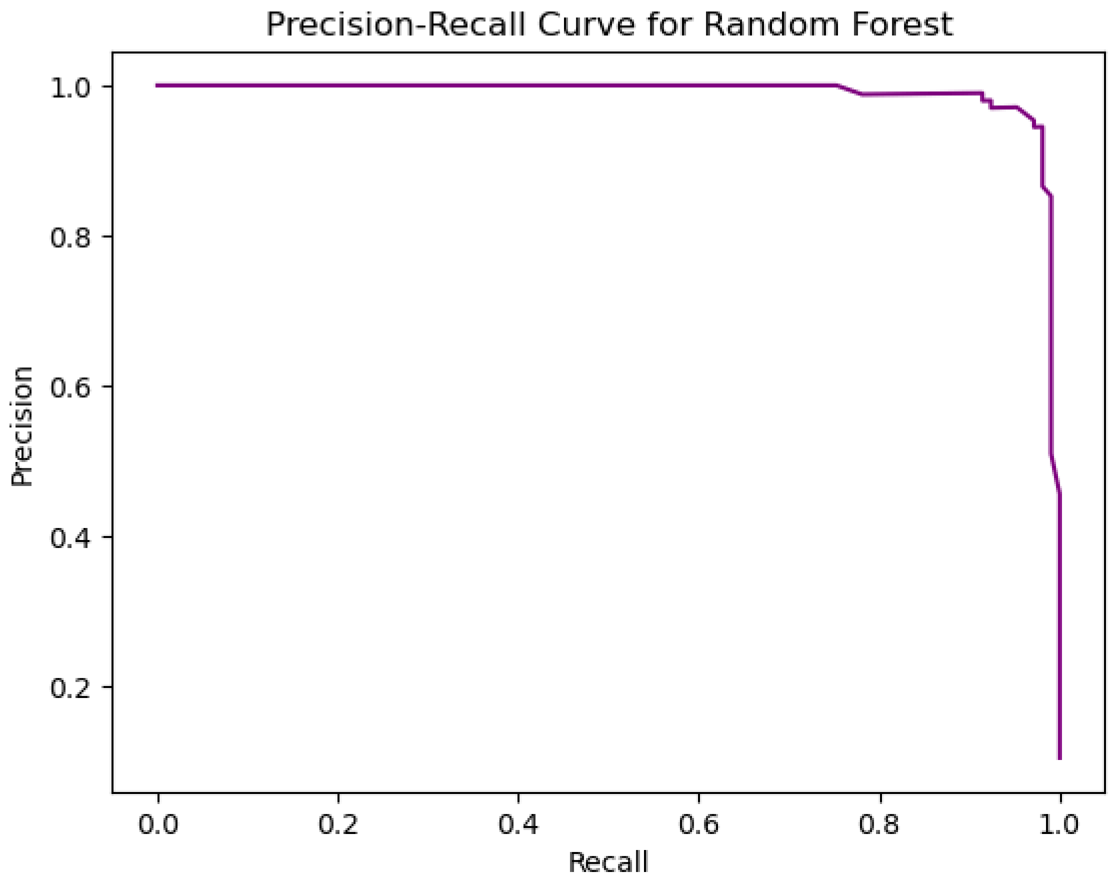
In [99]:

```
# Assuming y_pred contains the predicted labels and y_test contains the true labels
cm_rf = confusion_matrix(y_test, y_pred)

# Extract values for true positives, false positives, true negatives, and false neg
tn_rf, fp_rf, fn_rf, tp_rf = cm_rf.ravel()

# Display the confusion matrix
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf)
disp_rf.plot(cmap='Purples')

# Customize the plot with true positives, false positives, true negatives, and fals
plt.title('Confusion Matrix (Random Forest)')
plt.xlabel(f'Predicted Label\n\nTrue Positives: {tp_rf}\nFalse Positives: {fp_rf}')
plt.ylabel(f'True Label\n\nFalse Negatives: {fn_rf}\nTrue Negatives: {tn_rf}')
plt.show()
```



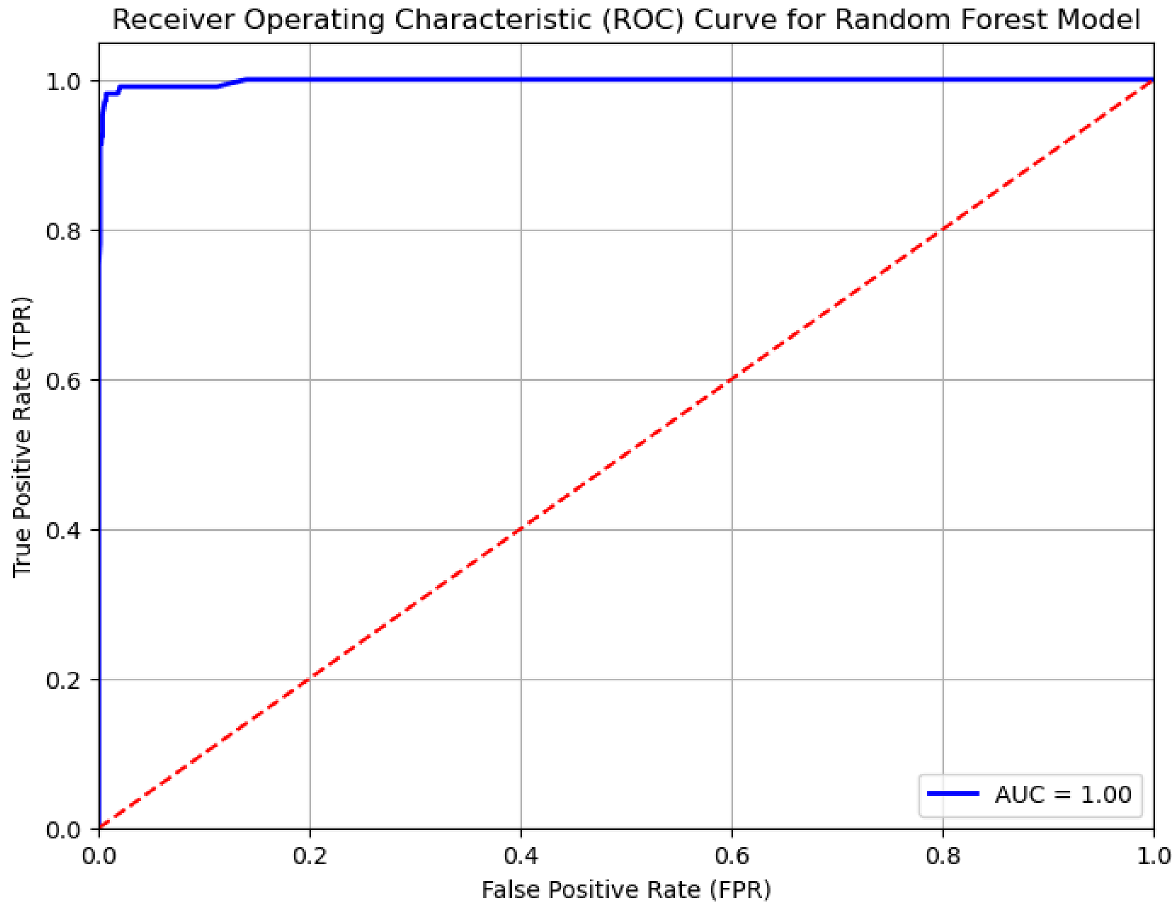

In [100…

```
best_rf = RandomForestClassifier(n_estimators=100, random_state=42)  # Instantiate
best_rf.fit(X_train, y_train)  # Train the model on your training data



# Calculate Precision and Recall
precision, recall, thresholds = precision_recall_curve(y_test, best_rf.predict_prob

# Create Precision-Recall Curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

# Adding axis labels and title to the plot
ax.set_title('Precision-Recall Curve for Random Forest')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

plt.savefig('Precision-Recall Curve for Random Forest.jpg')
plt.show()
```




In [101…

```
y_pred_proba_rf = model.predict_proba(X_test)[:, 1]  # Probability estimates of the
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_pred_proba_rf)

# Calculate the AUC score
auc_score_rf = roc_auc_score(y_test, y_pred_proba_rf)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='blue', lw=2, label=f'AUC = {auc_score_rf:.2f}')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line representing
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve for Random Forest Model')
plt.legend(loc='lower right')  # Display AUC score in the legend
plt.grid(True)
plt.show()
```




# knn

In [102…
```
# Initialize the KNN classifier
knn_model = KNeighborsClassifier()
```

In [103…
```
# Train the model on the training data
knn_model.fit(X_train, y_train)
```

Out[103]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [104…
```
#Evaluating the performance of the model using the test set
knn_Y_predict = knn_model.predict(X_test)
```

In [105…
```
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, knn_Y_predict))
```

```
Accuracy: 0.964
```

In [106…
```
# Print classification report
print("KNN Classification Report:")
print(classification_report(y_test,knn_Y_predict))
```

```
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       895
           1       0.85      0.80      0.82       105

    accuracy                           0.96      1000
   macro avg       0.91      0.89      0.90      1000
weighted avg       0.96      0.96      0.96      1000
```

# Optimising the KNN Model

In [107…

```
# Defining the range of values for n_neighbors
neighbors = np.arange(1, 36)
```

In [108…

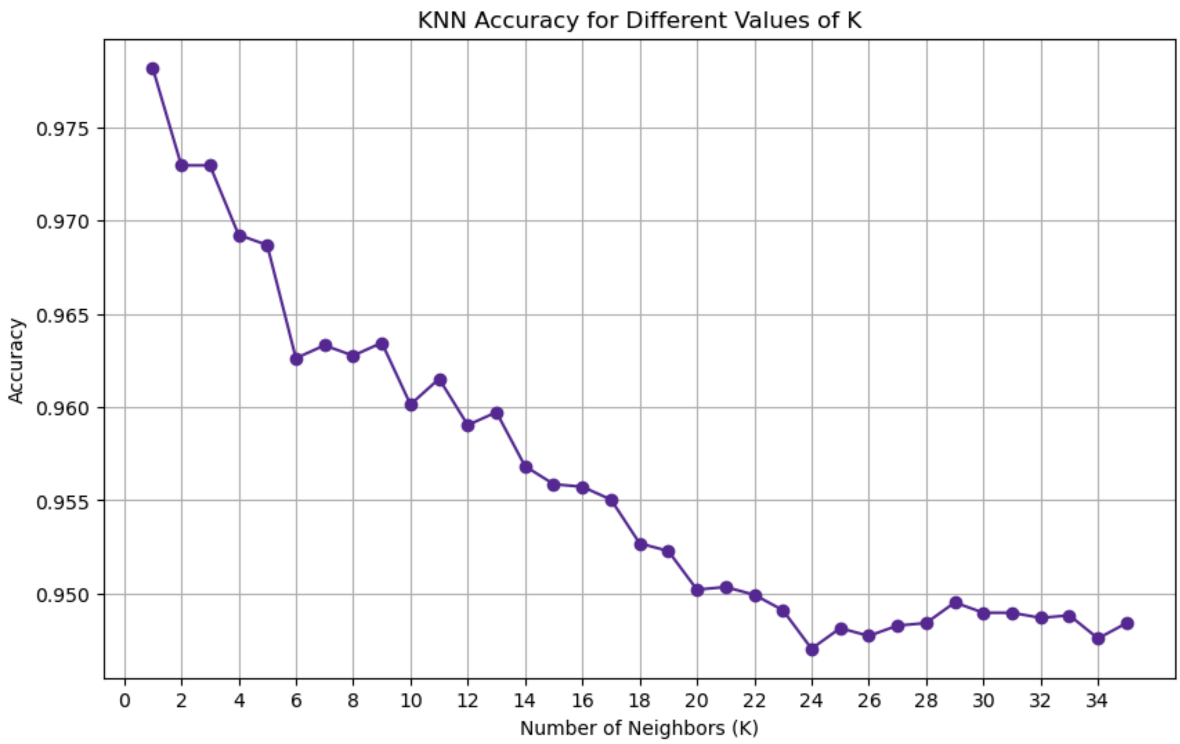
```
# Create an empty list to store the accuracy at different values of K
KNN_accuracy = []
```

In [109…

```
# Loop through each value of n_neighbors and perform grid search
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    grid_search = GridSearchCV(knn, {'n_neighbors': [k]}, cv=5)
    grid_search.fit(X_train, y_train)
    KNN_accuracy.append(grid_search.best_score_)
```

In [110…

```
# Plot the mean test scores
plt.figure(figsize=(10, 6))
plt.plot(neighbors, KNN_accuracy, marker='o')
plt.title('KNN Accuracy for Different Values of K')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.xticks(np.arange(0, 36, step=2))
plt.savefig('KNN Accuracy for Different Values of K.jpg')
plt.show()
```



In [111...

```
best_index = np.argmax(KNN_accuracy)
best_k = neighbors[best_index]

print(f"The best value of n_neighbors (K) is: {best_k}")
```

```
The best value of n_neighbors (K) is: 1
```

In [112...

```
knn= KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train,y_train)
best_k_predictions = knn.predict(X_test)
```

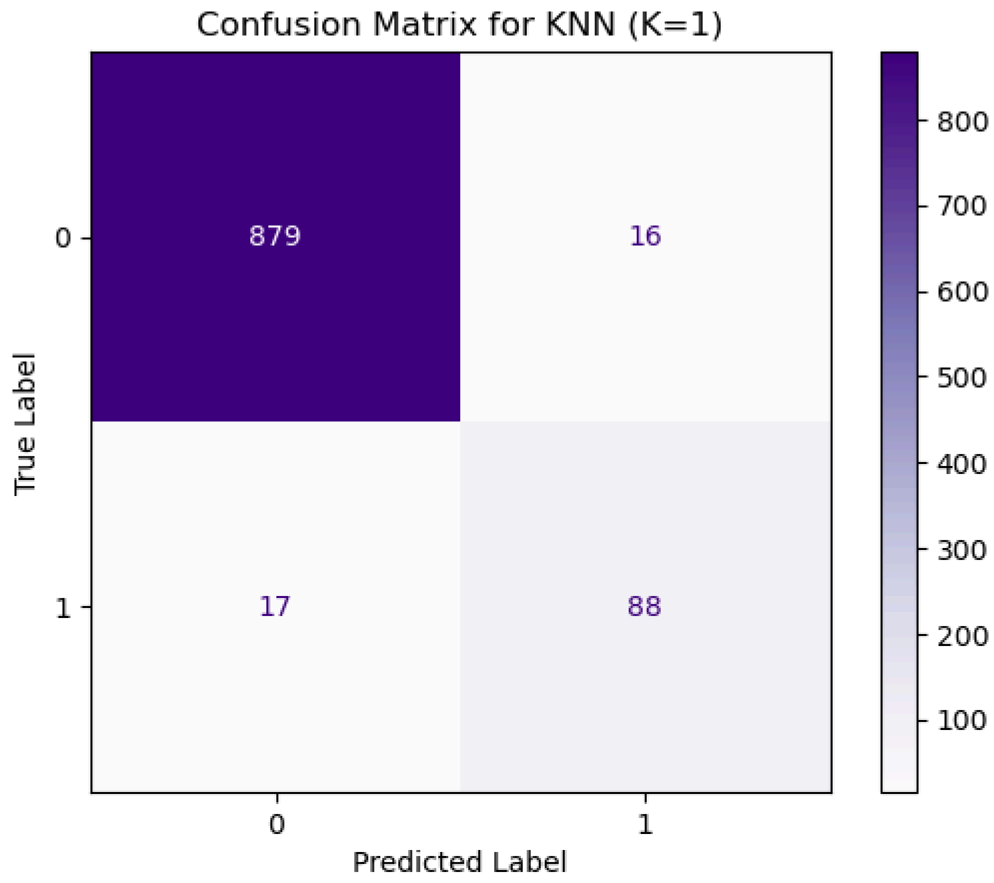
In [113...

```
# Calculate precision and recall for the best K value
accuracy_knn = accuracy_score(y_test, best_k_predictions)
precision_knn = precision_score(y_test, best_k_predictions)
recall_knn = recall_score(y_test, best_k_predictions)
f1_knn = f1_score(y_test, best_k_predictions)
# Print the accuracy, precision, and recall for the best K value
print(f"Best K value: {best_k}")
print(f"Accuracy for best K value: {accuracy_knn}")
print(f"Precision for best K value: {precision_knn}")
print(f"Recall for best K value: {recall_knn}")
print(f"F1 Score for best K value: {f1_knn}")

# Print classification report
print("KNN Classification Report:")
print(classification_report(y_test ,best_k_predictions))
```

```
Best K value: 1
Accuracy for best K value: 0.967
Precision for best K value: 0.8461538461538461
Recall for best K value: 0.8380952380952381
F1 Score for best K value: 0.8421052631578947
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       895
           1       0.85      0.84      0.84       105

    accuracy                           0.97      1000
   macro avg       0.91      0.91      0.91      1000
weighted avg       0.97      0.97      0.97      1000
```

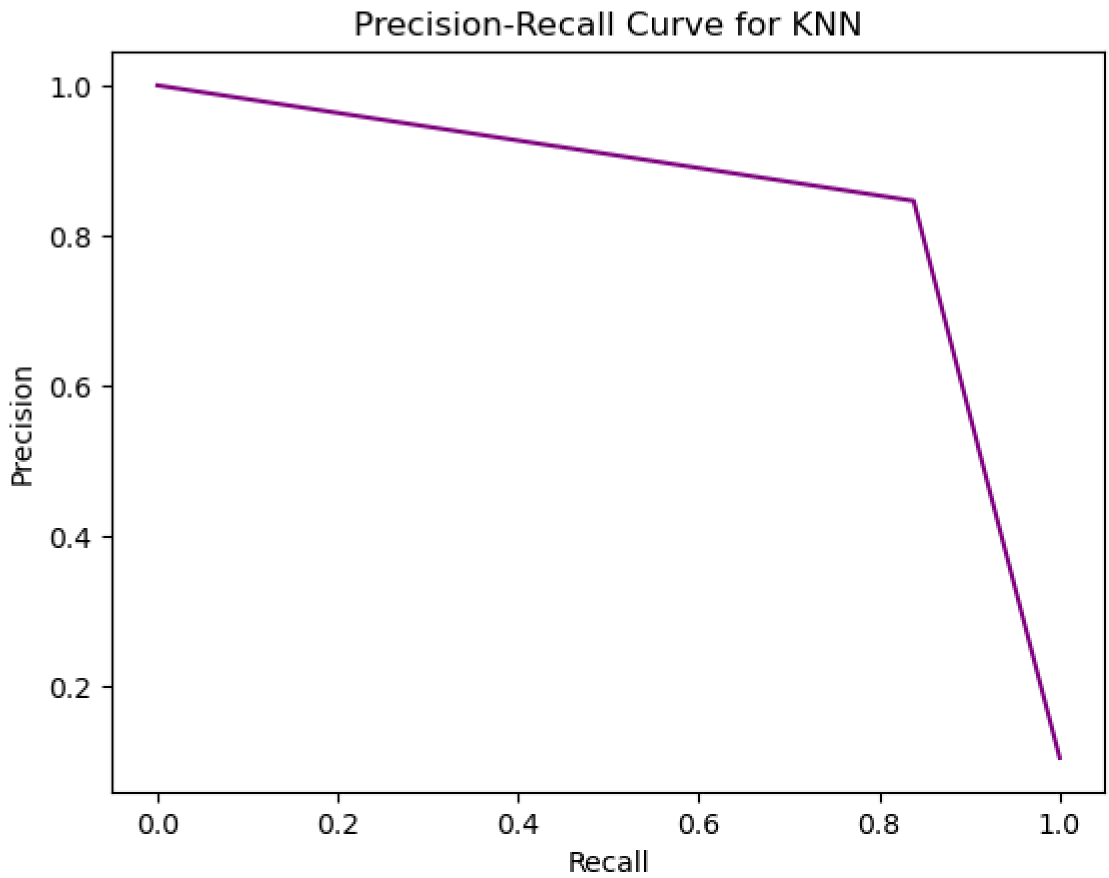
In [114...

```
# Plot the confusion matrix for the KNN Model
cm_knn = confusion_matrix(y_test, best_k_predictions)
disp_knn = metrics.ConfusionMatrixDisplay(confusion_matrix=cm_knn);
disp_knn.plot(cmap='Purples')
plt.title(f'Confusion Matrix for KNN (K={best_k})')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('Confusion Matrix for KNN.jpg')
plt.show()
```



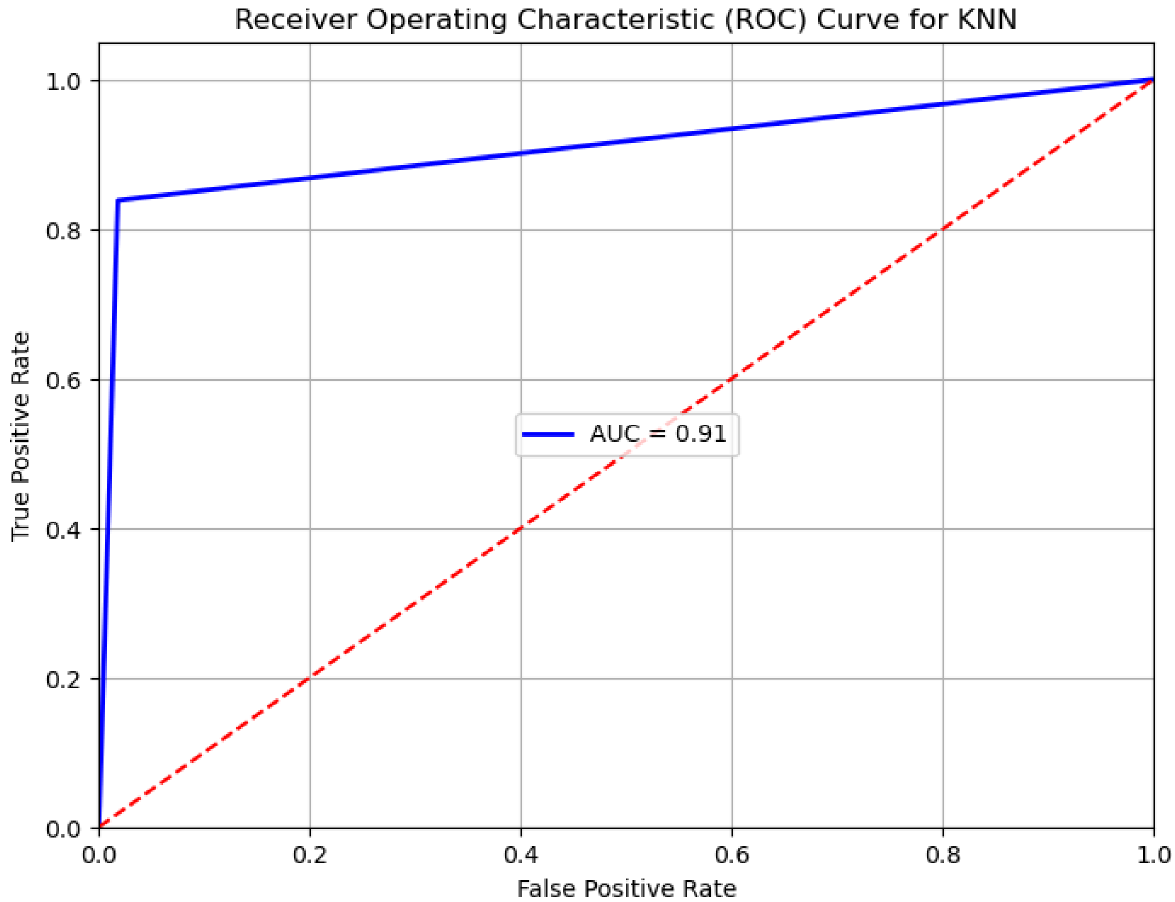

In [115…

```
# Calculate Precision and Recall
precision, recall, thresholds = precision_recall_curve(y_test, knn.predict_proba(X_
# Create Precision Recall Curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
# Adding axis labels and title to the plot
ax.set_title('Precision-Recall Curve for KNN')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
plt.savefig('Precision-Recall Curve for KNN.jpg')
plt.show()
```




In [116…

```
# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, best_k_predictions)
# Calculate the AUC score
auc_score = roc_auc_score(y_test, best_k_predictions)
# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {auc_score:.2f}')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for KNN')
plt.legend(loc='center')
plt.grid(True)
plt.savefig('Receiver Operating Characteristic (ROC) Curve for KNN.jpg')
plt.show()
```




In [117…

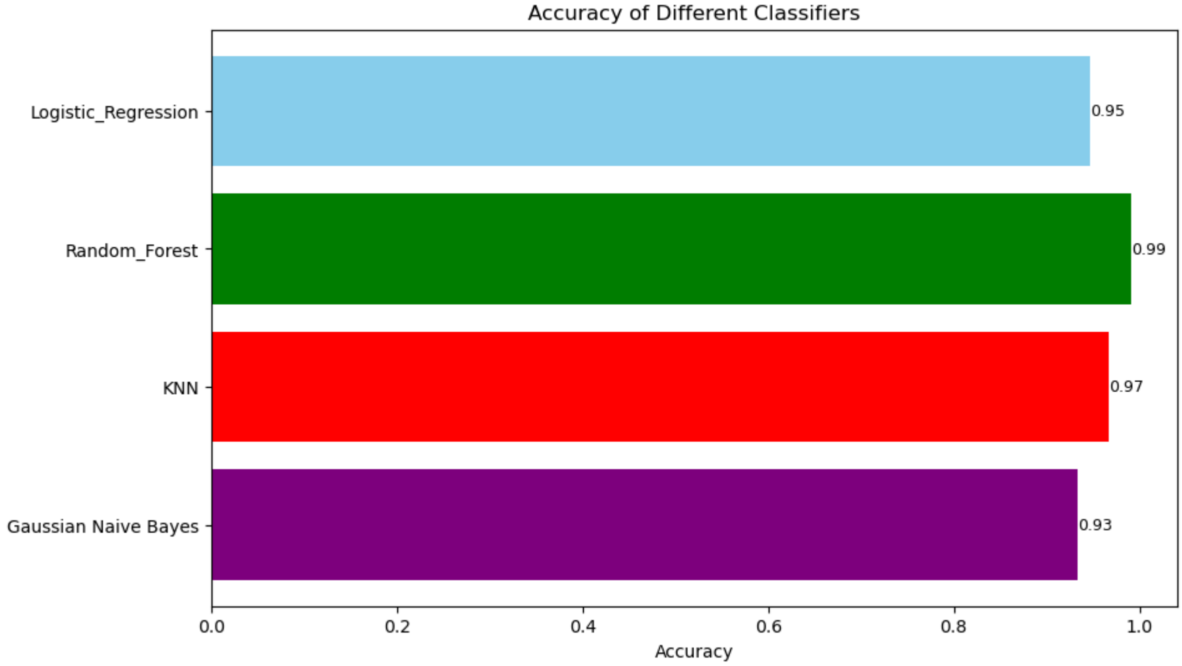
```
print(precision_rf)
print(precision_knn)
print(precision_nb)
```

```
0.9615384615384616
0.8461538461538461
[0.105      0.65853659 1.        ]
```

# Machine Learning Model Comparison

In [118…

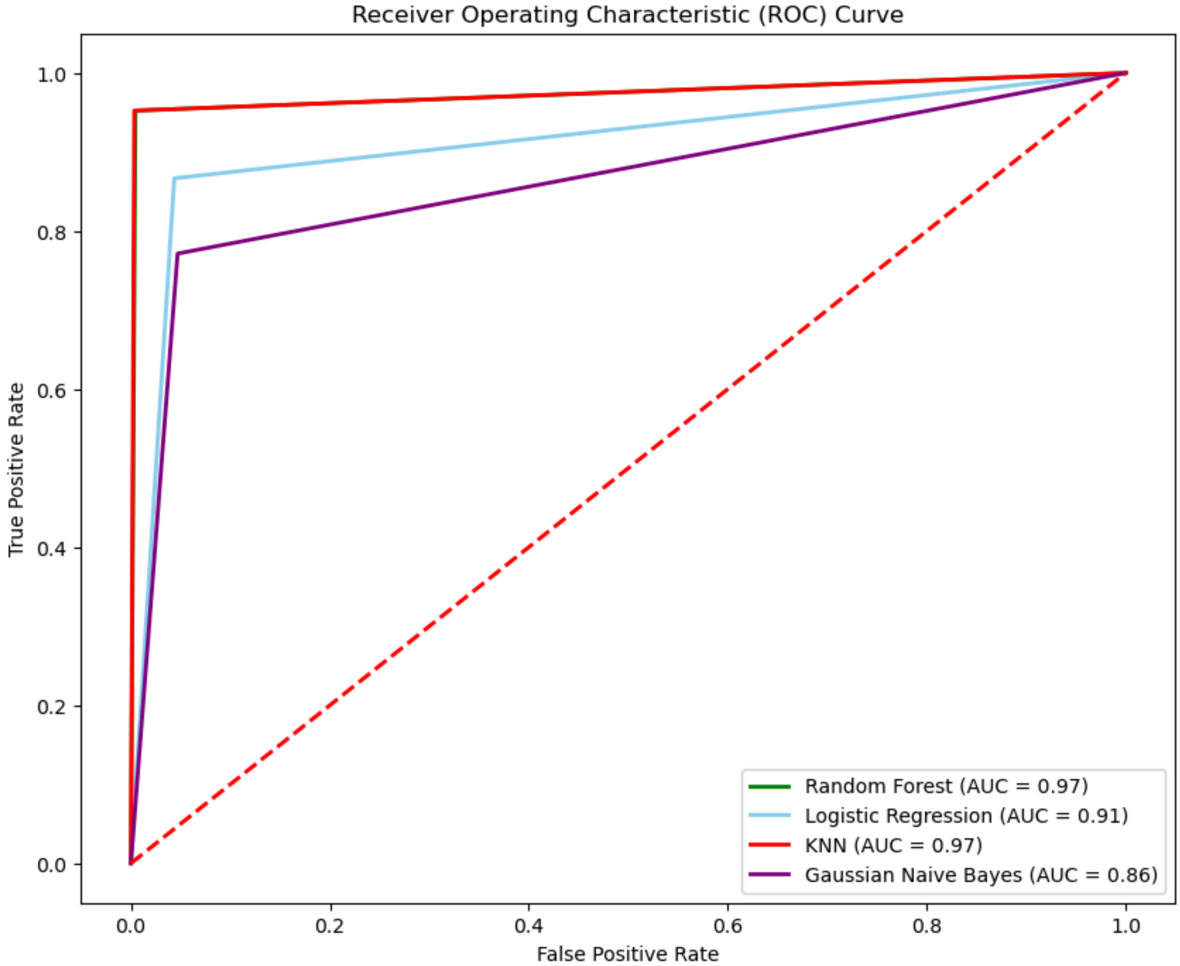
```
# Classifier names
classifiers = ['Logistic_Regression', 'Random_Forest', 'KNN', 'Gaussian Naive Bayes
# Accuracy scores obtained from cross-validation or testing
accuracy_scores = [accuracy_logreg, accuracy_rf, accuracy_knn, accuracy_nb]
# Define colors for each classifier
colors = ['skyblue', 'green', 'red', 'purple']
# Create a bar plot
plt.figure(figsize=(10, 6))
bars = plt.barh(classifiers, accuracy_scores, color=colors)
plt.xlabel('Accuracy')
plt.title('Accuracy of Different Classifiers')
plt.gca().invert_yaxis()
# Add labels on bars
for bar, score in zip(bars, accuracy_scores):
    plt.text(bar.get_width(), bar.get_y() + bar.get_height()/2, f'{score:.2f}',
             va='center', ha='left', fontsize=9)
plt.savefig('Model Comparision.jpg')
plt.show()
```




# Receiver Operating Characteristics (ROC)-Comparison Model

In [119…

```
# Calculate ROC curves and AUC scores for each model
models = {
    'Random Forest': (y_pred_rf, 'green'),
    'Logistic Regression': (y_pred_best, 'skyblue'),
    'KNN': (y_pred, 'red'),
    'Gaussian Naive Bayes': (y_pred_nb, 'purple')
}

plt.figure(figsize=(10, 8))

# Plot ROC curves for each model
for model_name, (y_pred, color) in models.items():
    fpr, tpr, _ = roc_curve(y_test, y_pred)
    auc_score = roc_auc_score(y_test, y_pred)  # Corrected here
    plt.plot(fpr, tpr, lw=2, label=f'{model_name} (AUC = {auc_score:.2f})', color=c

# Plot the diagonal line (no-skill line)
plt.plot([0, 1], [0, 1], linestyle='--', color='red', lw=2)

# Add legend, labels, and title
plt.legend(loc='lower right')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.savefig('Combined_ROC_Curve.png')
plt.show()
```




# Precision-Recall & F1 Comparison

In [120…

```
# Calculate average precision score for Naive Bayes
precision_nb_avg = np.mean(precision_nb)

# Calculate average recall score for Naive Bayes
recall_nb_avg = np.mean(recall_nb)

# Define the results dictionary
results = {
    'Model': ['Logistic Regression', 'Random Forest', 'KNN', 'Naive Bayes'],
    'Accuracy %': [accuracy_log, accuracy_rf, accuracy_knn, accuracy_nb],
    'Precision %': [precision_log * 100, precision_rf * 100, precision_knn * 100, p
    'Recall %': [recall_log * 100, recall_rf * 100, recall_knn * 100, recall_nb_avg
    'F1-Score': [ f1_log, f1_rf, f1_knn, f1_nb]
}

# Convert accuracy, precision, recall scores to percentages and F1 scores to string
results['Accuracy %'] = [f"{score:.2f}%" for score in results['Accuracy %']]
results['Precision %'] = [f"{score:.2f}%" for score in results['Precision %']]
results['Recall %'] = [f"{score:.2f}%" for score in results['Recall %']]
results['F1-Score'] = [f"{score:.2f}" for score in results['F1-Score']]

# Create a DataFrame
results_df = pd.DataFrame(results)

# Display the DataFrame
print(results_df)

# Plot the DataFrame as a table with increased font size
```

```
fig, ax = plt.subplots(figsize=(14, 8))
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=results_df.values, colLabels=results_df.columns, loc='cer

# Adjust font size
table.auto_set_font_size(False)
table.set_fontsize(10)

# Save the table as an image
plt.savefig('performance_metrics_table.jpg')
plt.show()
```

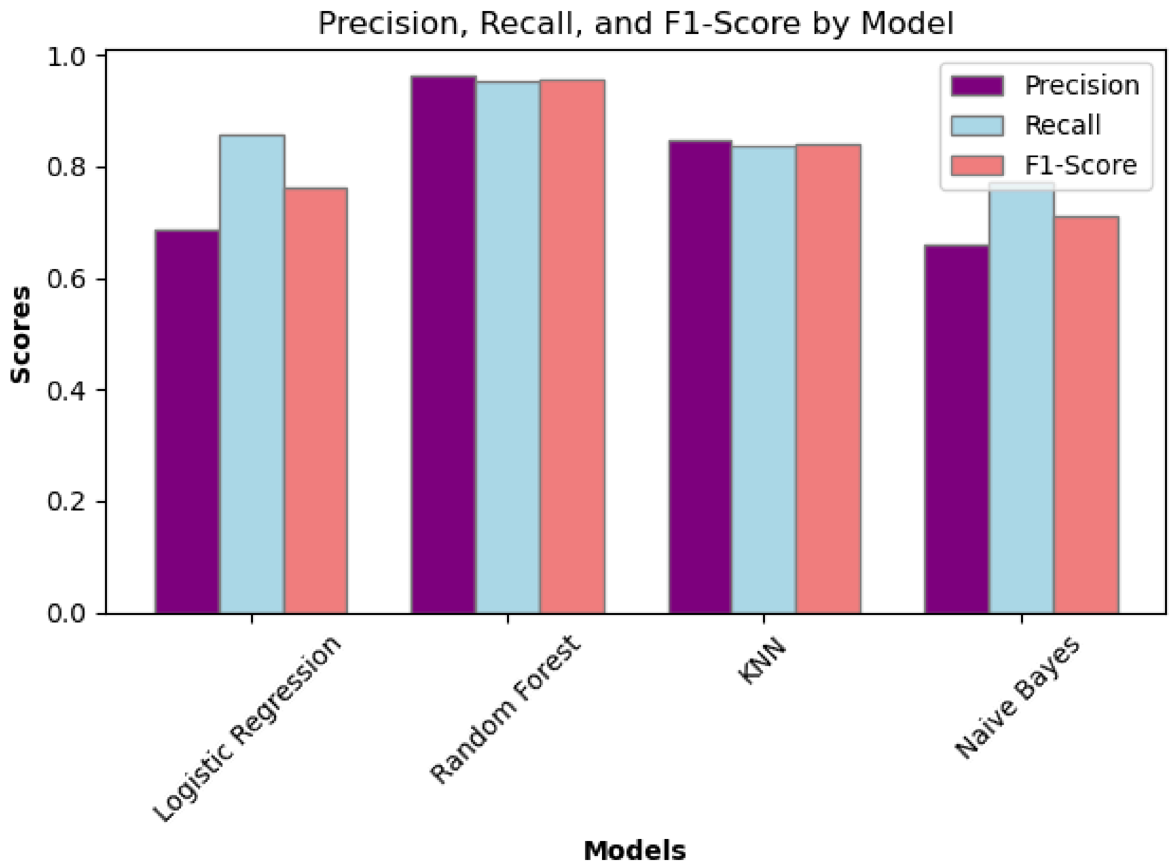
```
                 Model Accuracy % Precision % Recall % F1-Score
0  Logistic Regression      0.94%      68.70%   85.71%     0.76
1        Random Forest      0.99%      96.15%   95.24%     0.96
2                  KNN      0.97%      84.62%   83.81%     0.84
3          Naive Bayes      0.93%      58.78%   59.05%     0.71
```

| Model | Accuracy % | Precision % | Recall % | F1-Score |
|---|---|---|---|---|
| Logistic Regression | 0.94% | 68.70% | 85.71% | 0.76 |
| Random Forest | 0.99% | 96.15% | 95.24% | 0.96 |
| KNN | 0.97% | 84.62% | 83.81% | 0.84 |
| Naive Bayes | 0.93% | 58.78% | 59.05% | 0.71 |

In [121…

```
# Define data
models = ['Logistic Regression', 'Random Forest', 'KNN', 'Naive Bayes']
precision_scores = [precision_score(y_test, y_pred_log), precision_score(y_test, y_
recall_scores = [recall_score(y_test, y_pred_log), recall_score(y_test, y_pred_rf),
f1_scores = [f1_score(y_test, y_pred_log), f1_score(y_test, y_pred_rf), f1_score(y_

# Set the width of the bars
bar_width = 0.25
# Set the positions of the bars on the x-axis
r1 = np.arange(len(models))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]
# Plot grouped bar chart with custom colors
plt.bar(r1, precision_scores, color='purple', width=bar_width, edgecolor='grey', la
plt.bar(r2, recall_scores, color='lightblue', width=bar_width, edgecolor='grey', la
plt.bar(r3, f1_scores, color='lightcoral', width=bar_width, edgecolor='grey', label
# Add xticks on the middle of the group bars
plt.xlabel('Models', fontweight='bold')
plt.xticks([r + bar_width for r in range(len(models))], models, rotation=45)
# Add labels and title
plt.ylabel('Scores', fontweight='bold')
plt.title('Precision, Recall, and F1-Score by Model')
# Add legend
plt.legend()
```

```
# Show plot
plt.tight_layout()
plt.savefig('PRF.jpg')
plt.show()
```




In [ ]:

In [ ]: