GERMAN CREDIT RISK ANALYSIS

Contents

German Credit Dataset

1. Age (numeric)
2. Sex (text: male, female)
3. Job (numeric: 0 - unskilled and non-resident, 1 - unskilled and resident, 2 - skilled, 3 - highly skilled)
4. Housing (text: own, rent, or free)
5. Saving accounts (text - little, moderate, quite rich, rich)
6. Checking account (numeric, in DM - Deutsch Mark)
7. Credit amount (numeric, in DM)
8. Duration (numeric, in month)
9. Purpose (text: car, furniture/equipment, radio/TV, domestic appliances, repairs, education, business, vacation/others)

Import Libraries

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

Load 'german_credit_data.csv' and store it in a dataframe

In [2]:
```
df=pd.read_csv('german_credit_data.csv')
```

View top 5 rows

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purp
0	0	67	male	2	own	NaN	little	1169	6	radi
1	1	22	female	2	own	little	moderate	5951	48	radi
2	2	49	male	1	own	little	NaN	2096	12	educa
3	3	45	male	2	free	little	little	7882	42	furniture/equipr
4	4	53	male	2	free	little	little	4870	24	

Do data analysis with the dataset (shape of dataset,columns,null objects,unique elements)

Shape

In [4]:
```
df.shape
```

Out[4]:
```
(1000, 11)
```

Columns

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'Age', 'Sex', 'Job', 'Housing', 'Saving accounts',
       'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Risk'],
      dtype='object')
```

Number of null objects in each column

In [6]:
```
df.isna().sum()
```

Out[6]:
```
Unnamed: 0            0
Age                   0
Sex                   0
Job                   0
Housing               0
Saving accounts     183
Checking account    394
Credit amount         0
Duration              0
Purpose               0
Risk                  0
dtype: int64
```

In [7]:
```
df['Saving accounts'].unique()
```

Out[7]:
```
array([nan, 'little', 'quite rich', 'rich', 'moderate'], dtype=object)
```

In [8]:
```
df['Checking account'].unique()
```

Out[8]:
```
array(['little', 'moderate', nan, 'rich'], dtype=object)
```

Fill all null objects with 'Unknown'

In [9]:
```
df=df.fillna({'Saving accounts':'unknown','Checking account':'unknown'})
```

In [10]:
```
df.isna().sum()
```

Out[10]:
```
Unnamed: 0          0
Age                 0
Sex                 0
Job                 0
Housing             0
Saving accounts     0
Checking account    0
Credit amount       0
Duration            0
Purpose             0
Risk                0
dtype: int64
```

In [11]:
```
df['Saving accounts'].unique()
```

Out[11]:
```
array(['unknown', 'little', 'quite rich', 'rich', 'moderate'],
      dtype=object)
```

Unique elements in each column

In [12]:
```
df['Checking account'].unique()
```

Out[12]:
```
array(['little', 'moderate', 'unknown', 'rich'], dtype=object)
```

In [13]:
```
df.columns
```

Out[13]:
```
Index(['Unnamed: 0', 'Age', 'Sex', 'Job', 'Housing', 'Saving accounts',
       'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Risk'],
      dtype='object')
```

In [14]:
```
df['Credit amount'].unique()
```

Out[14]:
```
array([ 1169,  5951,  2096,  7882,  4870,  9055,  2835,  6948,  3059,
        5234,  1295,  4308,  1567,  1199,  1403,  1282,  2424,  8072,
       12579,  3430,  2134,  2647,  2241,  1804,  2069,  1374,   426,
         409,  2415,  6836,  1913,  4020,  5866,  1264,  1474,  4746,
        6110,  2100,  1225,   458,  2333,  1158,  6204,  6187,  6143,
        1393,  2299,  1352,  7228,  2073,  5965,  1262,  3378,  2225,
         783,  6468,  9566,  1961,  6229,  1391,  1537,  1953, 14421,
        3181,  5190,  2171,  1007,  1819,  2394,  8133,   730,  1164,
        5954,  1977,  1526,  3965,  4771,  9436,  3832,  5943,  1213,
        1568,  1755,  2315,  1412, 12612,  2249,  1108,   618,  1409,
         797,  3617,  1318, 15945,  2012,  2622,  2337,  7057,  1469,
        2323,   932,  1919,  2445, 11938,  6458,  6078,  7721,  1410,
        1449,   392,  6260,  7855,  1680,  3578,  7174,  2132,  4281,
        2366,  1835,  3868,  1768,   781,  1924,  2121,   701,   639,
        1860,  3499,  8487,  6887,  2708,  1984, 10144,  1240,  8613,
         766,  2728,  1881,   709,  4795,  3416,  2462,  2288,  3566,
         860,   682,  5371,  1582,  1346,  5848,  7758,  6967,  1288,
         339,  3512,  1898,  2872,  1055,  7308,   909,  2978,  1131,
        1577,  3972,  1935,   950,   763,  2064,  1414,  3414,  7485,
        2577,   338,  1963,   571,  9572,  4455,  1647,  3777,   884,
        1360,  5129,  1175,   674,  3244,  4591,  3844,  3915,  2108,
        3031,  1501,  1382,   951,  2760,  4297,   936,  1168,  5117,
         902,  1495, 10623,  1424,  6568,  1413,  3074,  3835,  5293,
        1908,  3342,  3104,  3913,  3021,  1364,   625,  1200,   707,
        4657,  2613, 10961,  7865,  1478,  3149,  4210,  2507,  2141,
         866,  1544,  1823, 14555,  2767,  1291,  2522,   915,  1595,
        4605,  1185,  3447,  1258,   717,  1204,  1925,   433,   666,
        2251,  2150,  4151,  2030,  7418,  2684,  2149,  3812,  1154,
        1657,  1603,  5302,  2748,  1231,   802,  6304,  1533,  8978,
         999,  2662,  1402, 12169,  3060, 11998,  2697,  2404,  4611,
        1901,  3368,  1574,  1445,  1520,  3878, 10722,  4788,  7582,
        1092,  1024,  1076,  9398,  6419,  4796,  7629,  9960,  4675,
        1287,  2515,  2745,   672,  3804,  1344,  1038, 10127,  1543,
        4811,   727,  1237,   276,  5381,  5511,  3749,   685,  1494,
        2746,   708,  4351,  3643,  4249,  1938,  2910,  2659,  1028,
        3398,  5801,  1525,  4473,  1068,  6615,  1864,  7408, 11590,
        4110,  3384,  2101,  1275,  4169,  1521,  5743,  3599,  3213,
        4439,  3949,  1459,   882,  3758,  1743,  1136,  1236,   959,
        3229,  6199,  1246,  2331,  4463,   776,  2406,  1239,  3399,
        2247,  1766,  2473,  1542,  3850,  3650,  3446,  3001,  3079,
        6070,  2146, 13756, 14782,  7685,  2320,   846, 14318,   362,
        2212, 12976,  1283,  1330,  4272,  2238,  1126,  7374,  2326,
        1820,   983,  3249,  1957, 11760,  2578,  2348,  1223,  1516,
        1473,  1887,  8648,  2899,  2039,  2197,  1053,  3235,   939,
        1967,  7253,  2292,  1597,  1381,  5842,  2579,  8471,  2782,
        1042,  3186,  2028,   958,  1591,  2762,  2779,  2743,  1149,
        1313,  1190,  3448, 11328,  1872,  2058,  2136,  1484,   660,
        3394,   609,  1884,  1620,  2629,   719,  5096,  1244,  1842,
        2576,  1512, 11054,   518,  2759,  2670,  4817,  2679,  3905,
        3386,   343,  4594,  3620,  1721,  3017,   754,  1950,  2924,
        1659,  7238,  2764,  4679,  3092,   448,   654,  1238,  1245,
        3114,  2569,  5152,  1037,  3573,  1201,  3622,   960,  1163,
        1209,  3077,  3757,  1418,  3518,  1934,  8318,   368,  2122,
        2996,  9034,  1585,  1301,  1323,  3123,  5493,  1216,  1207,
        1309,  2360,  6850,  8588,   759,  4686,  2687,   585,  2255,
        1361,  7127,  1203,   700,  5507,  3190,  7119,  3488,  1113,
        7966,  1532,  1503,  2302,   662,  2273,  2631,  1311,  3105,
```

```
 2319,  3612,  7763,  3049,  1534,  2032,  6350,  2864,  1255,
 1333,  2022,  1552,   626,  8858,   996,  1750,  6999,  1995,
 1331,  2278,  5003,  3552,  1928,  2964,  1546,   683, 12389,
 4712,  1553,  1372,  3979,  6758,  3234,  5433,   806,  1082,
 2788,  2930,  1927,  2820,   937,  1056,  3124,  1388,  2384,
 2133,  2799,  1289,  1217,  2246,   385,  1965,  1572,  2718,
 1358,   931,  1442,  4241,  2775,  3863,  2329,   918,  1837,
 3349,  2828,  4526,  2671,  2051,  1300,   741,  3357,  3632,
 1808, 12204,  9157,  3676,  3441,   640,  3652,  1530,  3914,
 1858,  2600,  1979,  2116,  1437,  4042,  3660,  1444,  1980,
 1355,  1376, 15653,  1493,  4370,   750,  1308,  4623,  1851,
 1880,  7980,  4583,  1386,   947,   684,  7476,  1922,  2303,
 8086,  2346,  3973,   888, 10222,  4221,  6361,  1297,   900,
 1050,  1047,  6314,  3496,  3609,  4843,  4139,  5742, 10366,
 2080,  2580,  4530,  5150,  5595,  1453,  1538,  2279,  5103,
 9857,  6527,  1347,  2862,  2753,  3651,   975,  2896,  4716,
 2284,  1103,   926,  1800,  1905,  1123,  6331,  1377,  2503,
 2528,  5324,  6560,  2969,  1206,  2118,   629,  1198,  2476,
 1138, 14027,  7596,  1505,  3148,  6148,  1337,  1228,   790,
 2570,   250,  1316,  1882,  6416,  6403,  1987,   760,  2603,
 3380,  3990, 11560,  4380,  6761,  4280,  2325,  1048,  3160,
 2483, 14179,  1797,  2511,  1274,  5248,  3029,   428,   976,
  841,  5771,  1555,  1285,  1299,  1271,   691,  5045,  2124,
 2214, 12680,  2463,  1155,  3108,  2901,  1655,  2812,  8065,
 3275,  2223,  1480,  1371,  3535,  3509,  5711,  3872,  4933,
 1940,   836,  1941,  2675,  2751,  6224,  5998,  1188,  6313,
 1221,  2892,  3062,  2301,  7511,  1549,  1795,  7472,  9271,
  590,   930,  9283,  1778,   907,   484,  9629,  3051,  3931,
 7432,  1338,  1554, 15857,  1345,  1101,  3016,  2712,   731,
 3780,  1602,  3966,  4165,  8335,  6681,  2375, 11816,  5084,
 2327,   886,   601,  2957,  2611,  5179,  2993,  1943,  1559,
 3422,  3976,  1249,  2235,  1471, 10875,   894,  3343,  3959,
 3577,  5804,  2169,  2439,  2210,  2221,  2389,  3331,  7409,
  652,  7678,  1343,   874,  3590,  1322,  3595,  1422,  6742,
 7814,  9277,  2181,  1098,  4057,   795,  2825, 15672,  6614,
 7824,  2442,  1829,  5800,  8947,  2606,  1592,  2186,  4153,
 2625,  3485, 10477,  1278,  1107,  3763,  3711,  3594,  3195,
 4454,  4736,  2991,  2142,  3161, 18424,  2848, 14896,  2359,
 3345,  1817, 12749,  1366,  2002,  6872,   697,  1049, 10297,
 1867,  1747,  1670,  1224,   522,  1498,   745,  2063,  6288,
 6842,  3527,   929,  1455,  1845,  8358,  2859,  3621,  2145,
 4113, 10974,  1893,  3656,  4006,  3069,  1740,  2353,  3556,
 2397,   454,  1715,  2520,  3568,  7166,  3939,  1514,  7393,
 1193,  7297,  2831,   753,  2427,  2538,  8386,  4844,  2923,
 8229,  1433,  6289,  6579,  3565,  1569,  1936,  2390,  1736,
 3857,   804,  4576], dtype=int64)
```

In [15]:
```
df['Duration'].unique()
```

Out[15]:
```
array([ 6, 48, 12, 42, 24, 36, 30, 15,  9, 10,  7, 60, 18, 45, 11, 27,  8,
       54, 20, 14, 33, 21, 16,  4, 47, 13, 22, 39, 28,  5, 26, 72, 40],
      dtype=int64)
```

In [16]:
```
df['Purpose'].unique()
```

Out[16]:
```
array(['radio/TV', 'education', 'furniture/equipment', 'car', 'business',
       'domestic appliances', 'repairs', 'vacation/others'], dtype=object)
```

In [17]:
```
df['Risk'].unique()
```

Out[17]:
```
array(['good', 'bad'], dtype=object)
```

In [18]:
```
df['Housing'].unique()
```

Out[18]:
```
array(['own', 'free', 'rent'], dtype=object)
```

In [19]:
```
for i in df.columns:
    print("unique values in", i ,"=",df[i].unique())
```

```
unique values in Unnamed: 0 = [  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  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  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  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
```

Create a New column Id by replacing unnamed column

In [20]:
```
df['Id']=df['Unnamed: 0']
```

In [21]:
```
df.head()
```

Out[21]:

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purp
0	0	67	male	2	own	unknown	little	1169	6	radi
1	1	22	female	2	own	little	moderate	5951	48	radi
2	2	49	male	1	own	little	unknown	2096	12	educa
3	3	45	male	2	free	little	little	7882	42	furniture/equipr
4	4	53	male	2	free	little	little	4870	24	

In [22]:
```
df=df.drop(columns=['Id'])
```

In [23]:
```
df.head()
```

Out[23]:

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purp
0	0	67	male	2	own	unknown	little	1169	6	radi
1	1	22	female	2	own	little	moderate	5951	48	radi
2	2	49	male	1	own	little	unknown	2096	12	educa
3	3	45	male	2	free	little	little	7882	42	furniture/equipr
4	4	53	male	2	free	little	little	4870	24	

In [24]:
```
df=df.rename(columns={'Unnamed: 0':'Id'})
```

In [25]:
```
df.head()
```

Out[25]:

	Id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Ri
0	0	67	male	2	own	unknown	little	1169	6	radio/TV	go
1	1	22	female	2	own	little	moderate	5951	48	radio/TV	b
2	2	49	male	1	own	little	unknown	2096	12	education	go
3	3	45	male	2	free	little	little	7882	42	furniture/equipment	go
4	4	53	male	2	free	little	little	4870	24	car	b

Find the information of the dataset

In [26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Id                1000 non-null   int64
 1   Age               1000 non-null   int64
 2   Sex               1000 non-null   object
 3   Job               1000 non-null   int64
 4   Housing           1000 non-null   object
 5   Saving accounts   1000 non-null   object
 6   Checking account  1000 non-null   object
 7   Credit amount     1000 non-null   int64
 8   Duration          1000 non-null   int64
 9   Purpose           1000 non-null   object
 10  Risk              1000 non-null   object
dtypes: int64(5), object(6)
memory usage: 86.1+ KB
```

Find basic statistical information about the dataset

In [27]:
```
df.describe()
```

Out[27]:

	Id	Age	Job	Credit amount	Duration
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	499.500000	35.546000	1.904000	3271.258000	20.903000
std	288.819436	11.375469	0.653614	2822.736876	12.058814
min	0.000000	19.000000	0.000000	250.000000	4.000000
25%	249.750000	27.000000	2.000000	1365.500000	12.000000
50%	499.500000	33.000000	2.000000	2319.500000	18.000000
75%	749.250000	42.000000	2.000000	3972.250000	24.000000
max	999.000000	75.000000	3.000000	18424.000000	72.000000

Fetch the following groupby results

In [28]:
```
df.groupby(['Sex','Risk','Purpose']).count()['Id']
```

Out[28]:
```
Sex     Risk  Purpose
female  bad   business                 7
              car                     40
              domestic appliances      2
              education                9
              furniture/equipment     28
              radio/TV                19
              repairs                  2
              vacation/others          2
        good  business                12
              car                     54
              domestic appliances      4
              education               15
              furniture/equipment     46
              radio/TV                66
              repairs                  3
              vacation/others          1
male    bad   business                27
              car                     66
              domestic appliances      2
              education               14
              furniture/equipment     30
              radio/TV                43
              repairs                  6
              vacation/others          3
        good  business                51
              car                    177
              domestic appliances      4
              education               21
              furniture/equipment     77
              radio/TV               152
              repairs                 11
              vacation/others          6
Name: Id, dtype: int64
```

In [29]:
```
df.groupby(['Purpose','Risk','Sex']).count()['Id']
```

Out[29]:

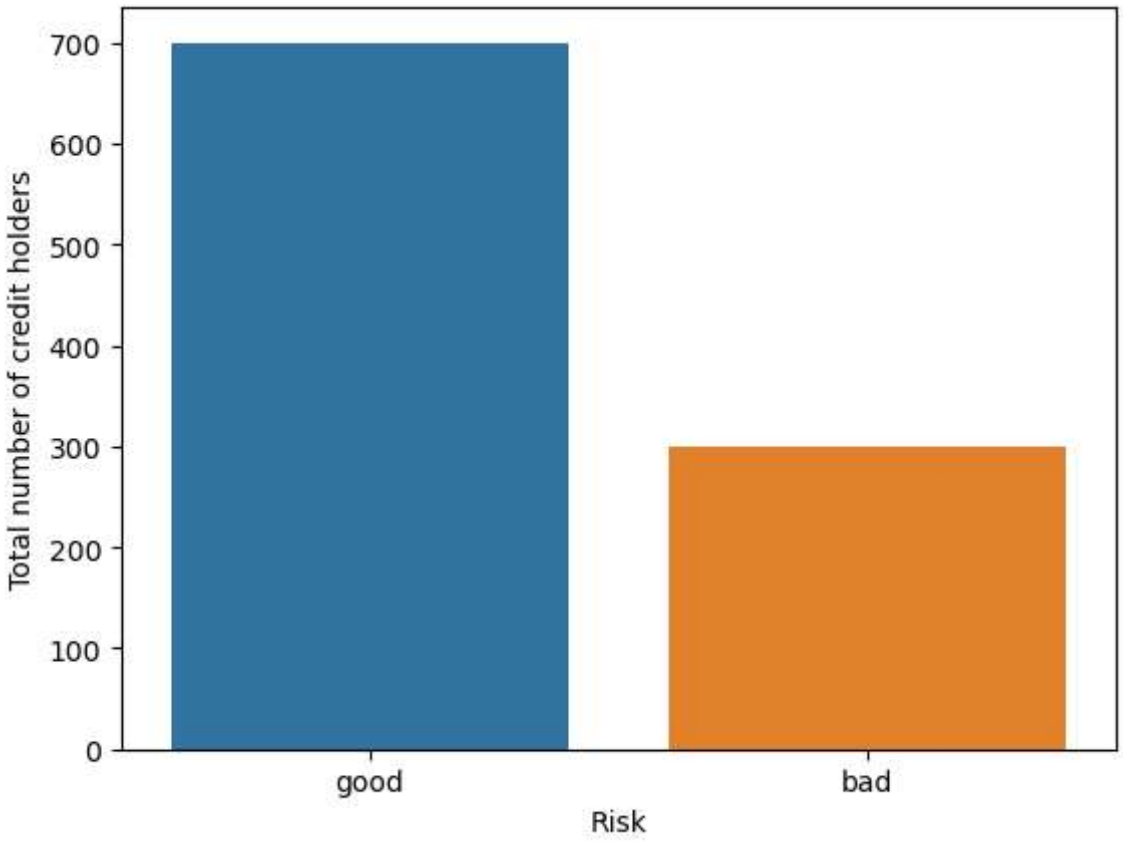
Purpose	Risk	Sex	
business	bad	female	7
		male	27
	good	female	12
		male	51
car	bad	female	40
		male	66
	good	female	54
		male	177
domestic appliances	bad	female	2
		male	2
	good	female	4
		male	4
education	bad	female	9
		male	14
	good	female	15
		male	21
furniture/equipment	bad	female	28
		male	30
	good	female	46
		male	77
radio/TV	bad	female	19
		male	43
	good	female	66
		male	152
repairs	bad	female	2
		male	6
	good	female	3
		male	11
vacation/others	bad	female	2
		male	3
	good	female	1
		male	6

Name: Id, dtype: int64

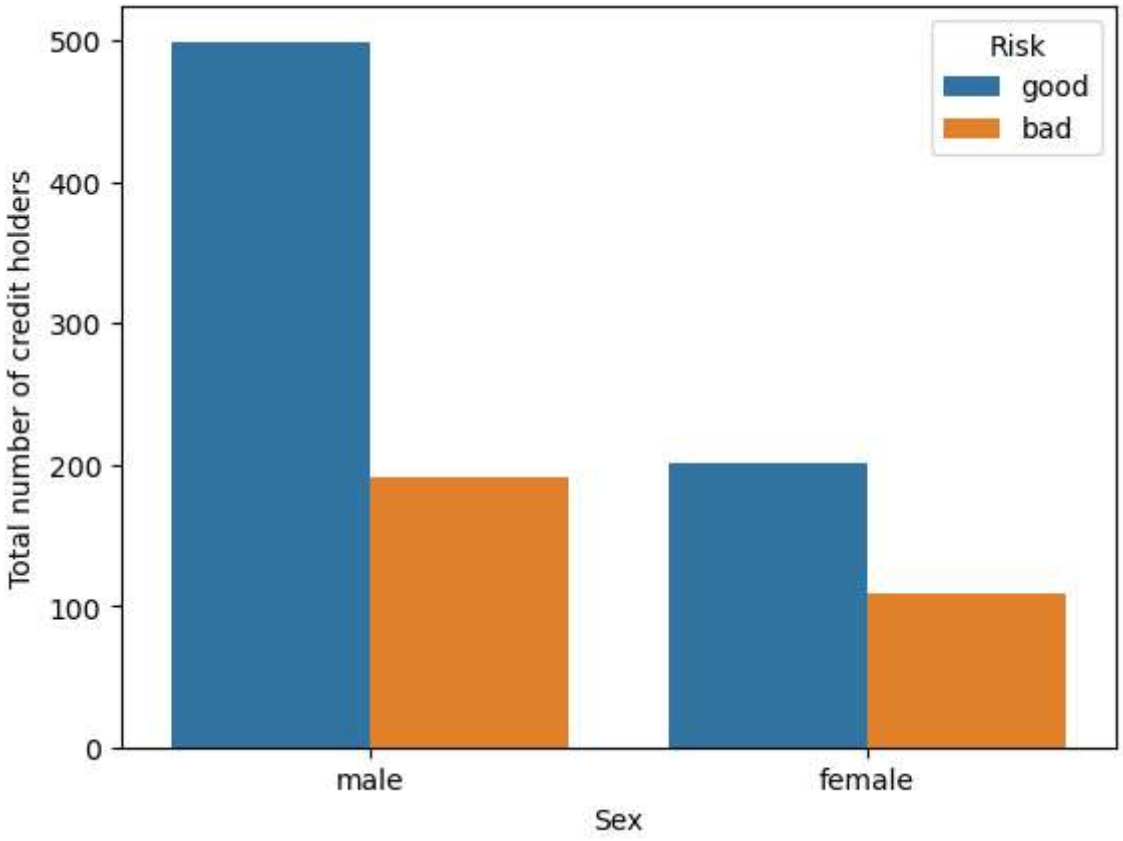
Data Visualization

Plot the graphs shown below

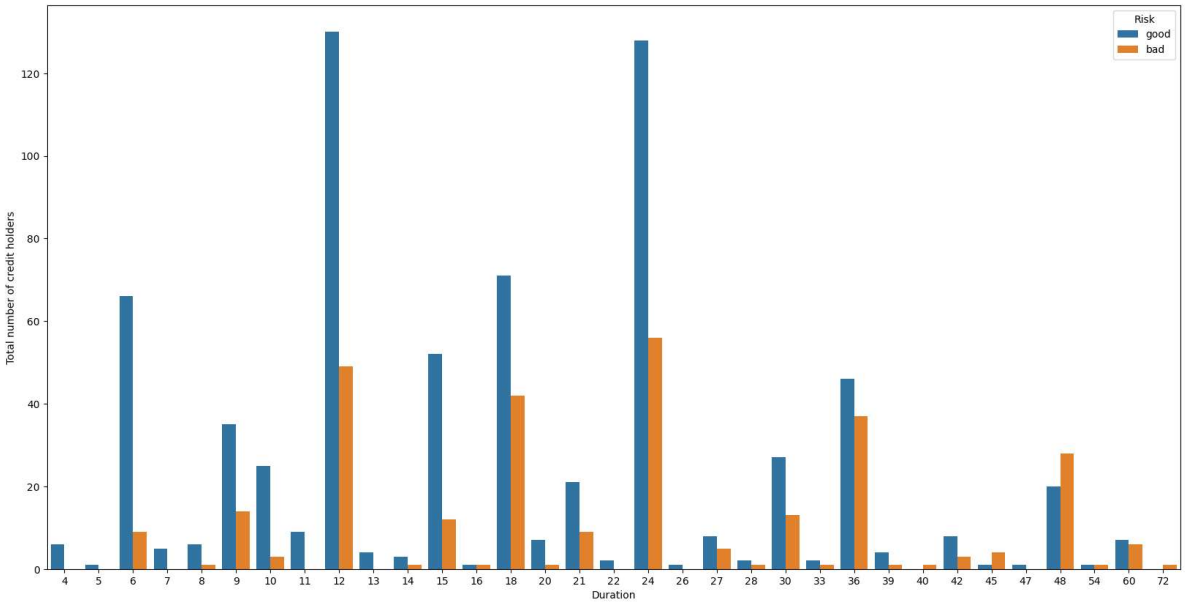
In [30]:
```
sns.countplot(x=df['Risk'])
plt.ylabel('Total number of credit holders')
plt.show()
```



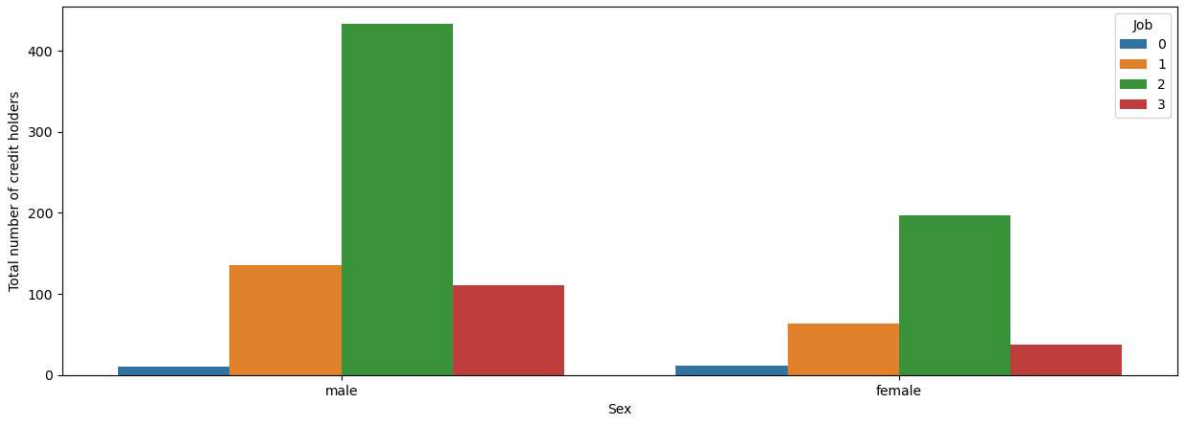
In [31]:
```
sns.countplot(x=df['Sex'],hue=df['Risk'])
plt.ylabel('Total number of credit holders')
plt.show()
```



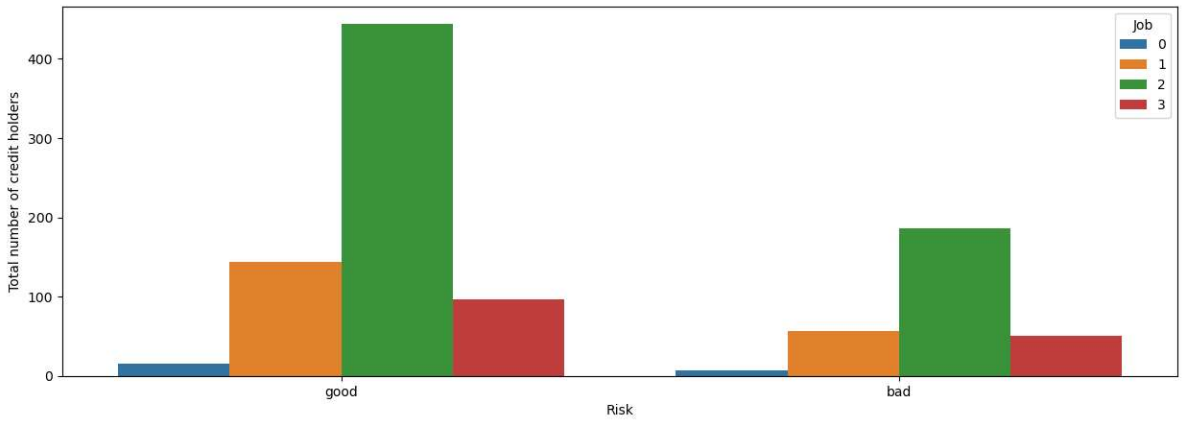
In [32]:
```
plt.figure(figsize=(20,10))
sns.countplot(x=df['Duration'],hue=df['Risk'])
plt.ylabel('Total number of credit holders')
plt.show()
```



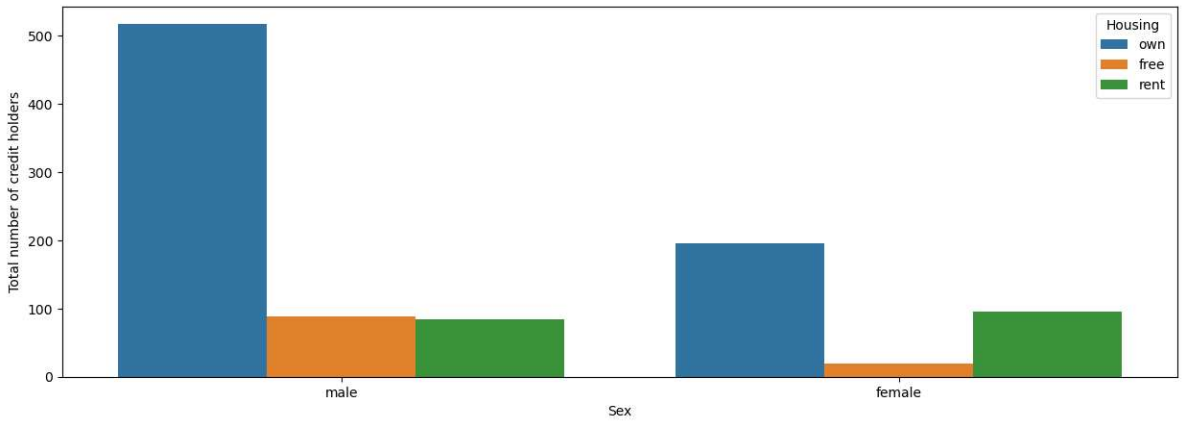
In [33]:
```
plt.figure(figsize=(15,5))
sns.countplot(x=df['Sex'],hue=df['Job'])
plt.ylabel('Total number of credit holders')
plt.show()
```



In [34]:
```
plt.figure(figsize=(15,5))
sns.countplot(x=df['Risk'],hue=df['Job'])
plt.ylabel('Total number of credit holders')
plt.show()
```



In [35]:
```
plt.figure(figsize=(15,5))
sns.countplot(x=df['Sex'],hue=df['Housing'])
plt.ylabel('Total number of credit holders')
plt.show()
```



In [36]:
```
df['Purpose'].unique()
```

Out[36]:
```
array(['radio/TV', 'education', 'furniture/equipment', 'car', 'business',
       'domestic appliances', 'repairs', 'vacation/others'], dtype=object)
```

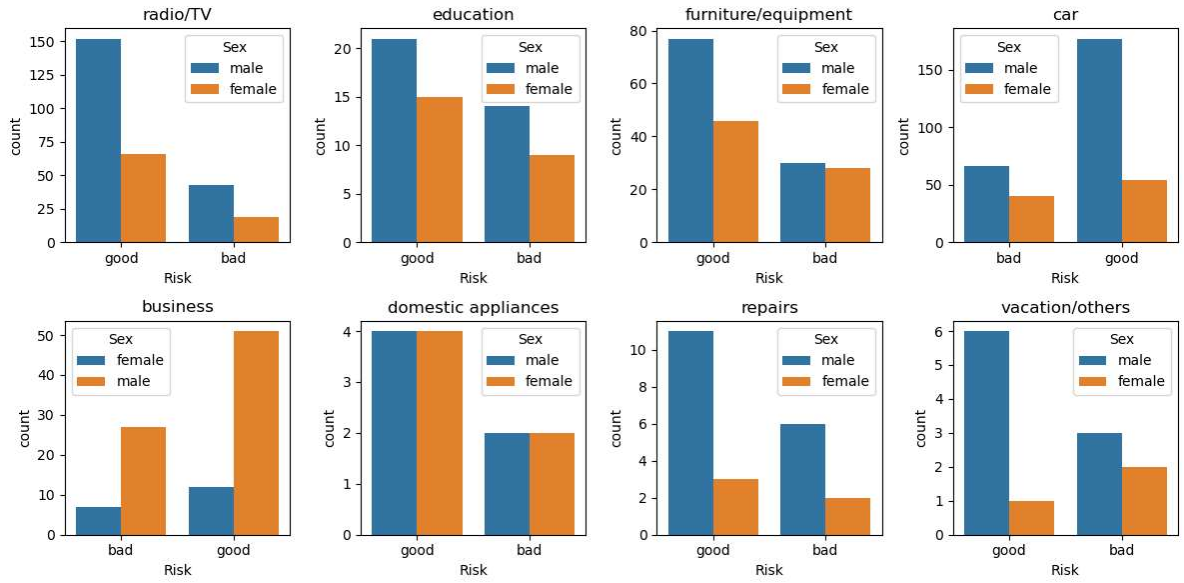
In [37]:
```
df[df['Purpose']=='radio/TV']
```

Out[37]:

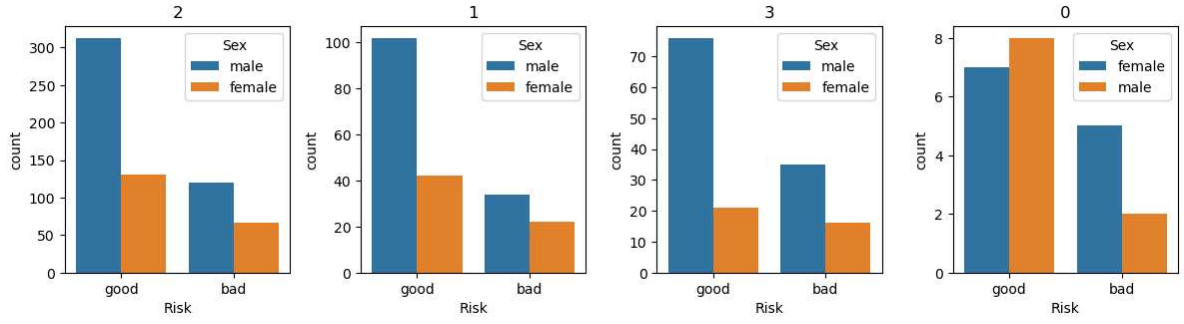
	Id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Risk
0	0	67	male	2	own	unknown	little	1169	6	radio/TV	good
1	1	22	female	2	own	little	moderate	5951	48	radio/TV	bad
8	8	61	male	1	own	rich	unknown	3059	12	radio/TV	good
12	12	22	female	2	own	little	moderate	1567	12	radio/TV	good
15	15	32	female	1	own	moderate	little	1282	24	radio/TV	bad
...	...	...	...	...	...	...	...	...	...	...	...
989	989	48	male	1	own	little	moderate	1743	24	radio/TV	good
991	991	34	male	1	own	moderate	unknown	1569	15	radio/TV	good
992	992	23	male	1	rent	unknown	little	1936	18	radio/TV	good
997	997	38	male	2	own	little	unknown	804	12	radio/TV	good
998	998	23	male	2	free	little	little	1845	45	radio/TV	bad

280 rows × 11 columns

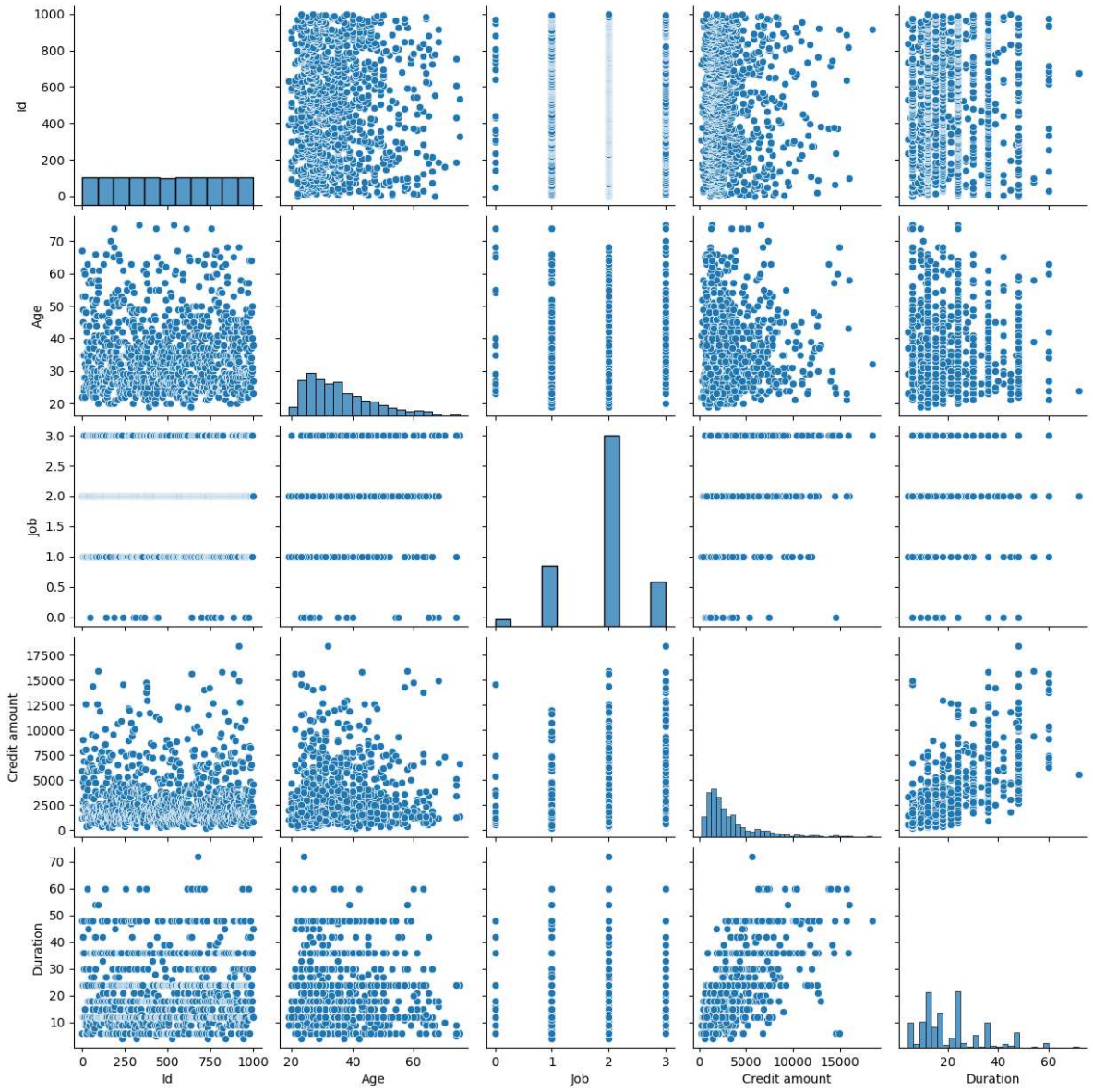
In [38]:
```
plt.figure(figsize = (12,6))
purpose = df.set_index('Purpose')
n = 1

for i in df['Purpose'].unique().tolist():
    plt.subplot(2,4,n)
    ax = sns.countplot(x = 'Risk', hue = 'Sex', data = purpose.loc[i])
    ax.set_title(i)
    n += 1
plt.tight_layout()
```



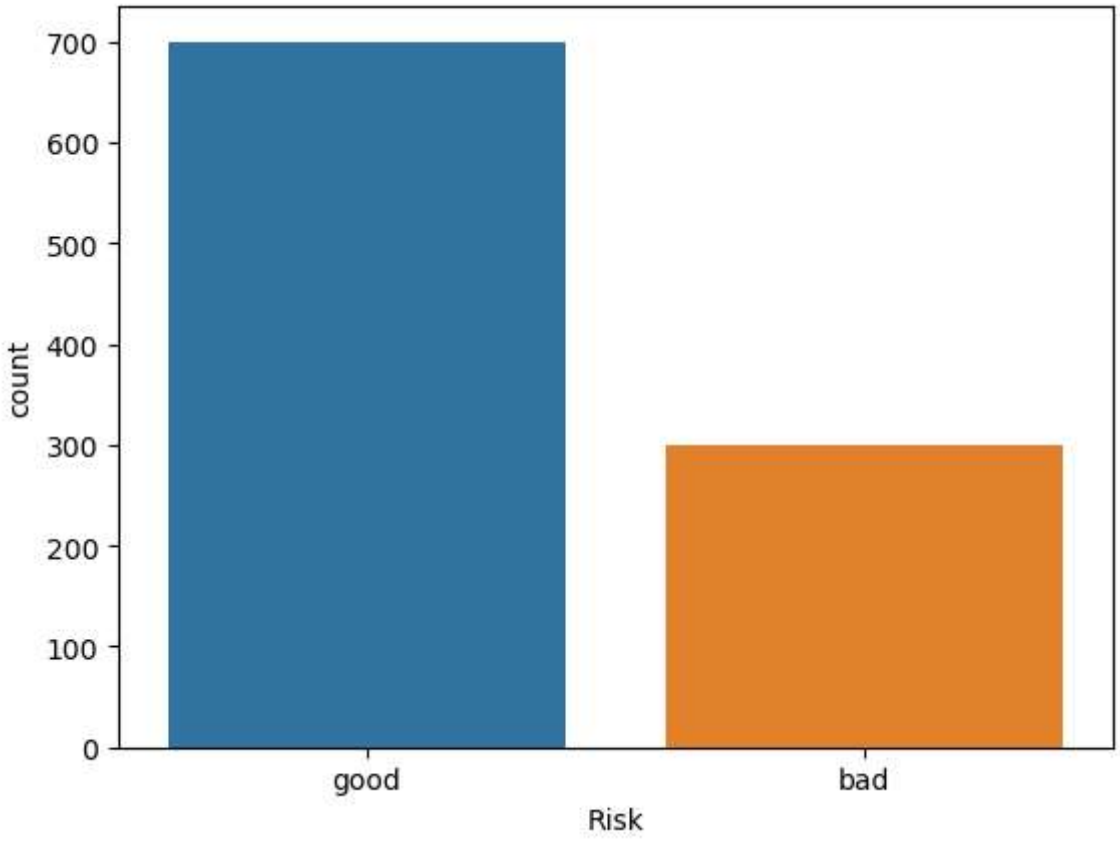
In [39]:
```
plt.figure(figsize = (12,6))
job= df.set_index('Job')
n = 1

for i in df['Job'].unique().tolist():
    plt.subplot(2,4,n)
    ax = sns.countplot(x = 'Risk', hue = 'Sex', data = job.loc[i])
    ax.set_title(i)
    n += 1
plt.tight_layout()
```



In [40]:
```
sns.pairplot(df)
plt.show()
```



In [41]:
```
sns.countplot(x='Risk', data=df)
plt.show()
```



Preprocess the dataset:

We have to convert all categorical values into numerical values.

Convert purpose column using Dummy variables

In [42]:
```
df.columns
```

Out[42]:
```
Index(['Id', 'Age', 'Sex', 'Job', 'Housing', 'Saving accounts',
       'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Risk'],
      dtype='object')
```

In [43]:
```
df.head()
```

Out[43]:

	Id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Ri
0	0	67	male	2	own	unknown	little	1169	6	radio/TV	go
1	1	22	female	2	own	little	moderate	5951	48	radio/TV	b
2	2	49	male	1	own	little	unknown	2096	12	education	go
3	3	45	male	2	free	little	little	7882	42	furniture/equipment	go
4	4	53	male	2	free	little	little	4870	24	car	b

In [44]:
```
purpose=pd.get_dummies(df['Purpose'])
newdf=pd.concat([df,purpose],axis='columns')
```

In [45]:
```
newdf.head()
```

Out[45]:

	Id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Ri
0	0	67	male	2	own	unknown	little	1169	6	radio/TV	go
1	1	22	female	2	own	little	moderate	5951	48	radio/TV	b
2	2	49	male	1	own	little	unknown	2096	12	education	go
3	3	45	male	2	free	little	little	7882	42	furniture/equipment	go
4	4	53	male	2	free	little	little	4870	24	car	b

In [46]:
```
newdf.columns
```

Out[46]:
```
Index(['Id', 'Age', 'Sex', 'Job', 'Housing', 'Saving accounts',
       'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Risk',
       'business', 'car', 'domestic appliances', 'education',
       'furniture/equipment', 'radio/TV', 'repairs', 'vacation/others'],
      dtype='object')
```

Use label encoding for rest of the categorical data

In [47]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

newdf['Sex']=le.fit_transform(df['Sex'])
newdf['Housing']=le.fit_transform(df['Housing'])
newdf['Saving accounts'] = le.fit_transform(df['Saving accounts'])
newdf['Checking account'] = le.fit_transform(df['Checking account'])
newdf['Risk'] = le.fit_transform(df['Risk'])
```

In [48]:
```
newdf.drop(columns=['Id','Purpose','car'],inplace=True)
```

In [49]:
```
newdf.head()
```

Out[49]:

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Risk	business	domestic appliances
0	67	1	2	1	4	0	1169	6	1	0	0
1	22	0	2	1	0	1	5951	48	0	0	0
2	49	1	1	1	0	3	2096	12	1	0	0
3	45	1	2	0	0	0	7882	42	1	0	0
4	53	1	2	0	0	0	4870	24	0	0	0

Check the unique values in the new dataframe

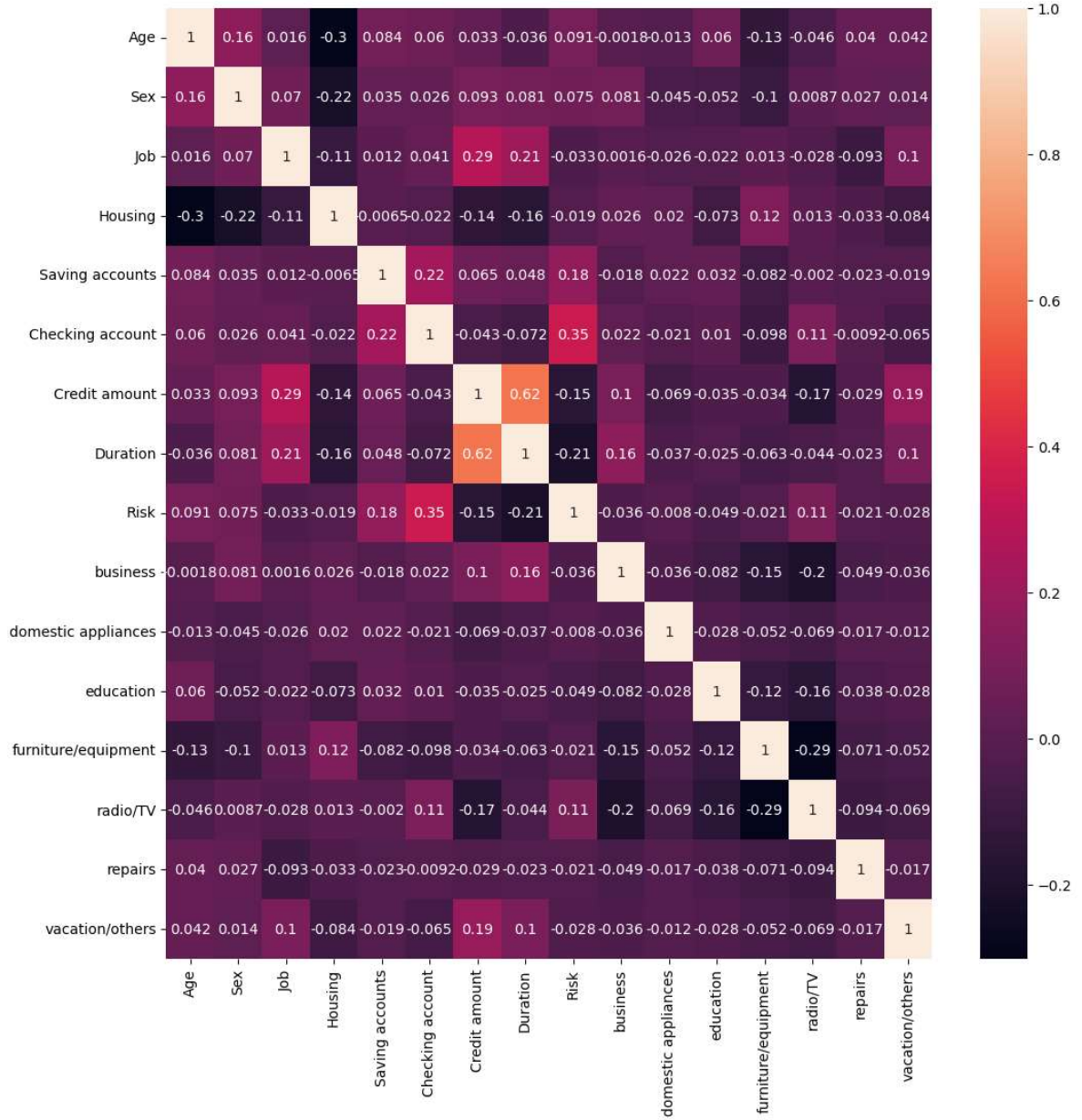
In [50]:
```
for i in newdf.columns:
    print("Unique values in ",i, "are :",newdf[i].unique())
```

```
Unique values in  Age are : [67 22 49 45 53 35 61 28 25 24 60 32 44 31 48 26
36 39 42 34 63 27 30 57
 33 37 58 23 29 52 50 46 51 41 40 66 47 56 54 20 21 38 70 65 74 68 43 55
 64 75 19 62 59]
Unique values in  Sex are : [1 0]
Unique values in  Job are : [2 1 3 0]
Unique values in  Housing are : [1 0 2]
Unique values in  Saving accounts are : [4 0 2 3 1]
Unique values in  Checking account are : [0 1 3 2]
Unique values in  Credit amount are : [ 1169  5951  2096  7882  4870  9055  2
835  6948  3059  5234  1295  4308
  1567  1199  1403  1282  2424  8072 12579  3430  2134  2647  2241  1804
  2069  1374   426   409  2415  6836  1913  4020  5866  1264  1474  4746
  6110  2100  1225   458  2333  1158  6204  6187  6143  1393  2299  1352
  7228  2073  5965  1262  3378  2225   783  6468  9566  1961  6229  1391
  1537  1953 14421  3181  5190  2171  1007  1819  2394  8133   730  1164
  5954  1977  1526  3965  4771  9436  3832  5943  1213  1568  1755  2315
  1412 12612  2249  1108   618  1409   797  3617  1318 15945  2012  2622
  2337  7057  1469  2323   932  1919  2445 11938  6458  6078  7721  1410
  1449   392  6260  7855  1680  3578  7174  2132  4281  2366  1835  3868
  1768   781  1924  2121   701   639  1860  3499  8487  6887  2708  1984
 10144  1240  8613   766  2728  1881   709  4795  3416  2462  2288  3566
   860   682  5371  1582  1346  5848  7758  6967  1288   339  3512  1898
  2872  1055  7308   909  2978  1131  1577  3972  1935   950   763  2064
  1414  3414  7485  2577   338  1963   571  9572  4455  1647  3777   884
  1360  5129  1175   674  3244  4591  3844  3915  2108  3031  1501  1382
   951  2760  4297   936  1168  5117   902  1495 10623  1424  6568  1413
  3074  3835  5293  1908  3342  3104  3913  3021  1364   625  1200   707
  4657  2613 10961  7865  1478  3149  4210  2507  2141   866  1544  1823
 14555  2767  1291  2522   915  1595  4605  1185  3447  1258   717  1204
  1925   433   666  2251  2150  4151  2030  7418  2684  2149  3812  1154
  1657  1603  5302  2748  1231   802  6304  1533  8978   999  2662  1402
 12169  3060 11998  2697  2404  4611  1901  3368  1574  1445  1520  3878
 10722  4788  7582  1092  1024  1076  9398  6419  4796  7629  9960  4675
  1287  2515  2745   672  3804  1344  1038 10127  1543  4811   727  1237
   276  5381  5511  3749   685  1494  2746   708  4351  3643  4249  1938
  2910  2659  1028  3398  5801  1525  4473  1068  6615  1864  7408 11590
  4110  3384  2101  1275  4169  1521  5743  3599  3213  4439  3949  1459
   882  3758  1743  1136  1236   959  3229  6199  1246  2331  4463   776
  2406  1239  3399  2247  1766  2473  1542  3850  3650  3446  3001  3079
  6070  2146 13756 14782  7685  2320   846 14318   362  2212 12976  1283
  1330  4272  2238  1126  7374  2326  1820   983  3249  1957 11760  2578
  2348  1223  1516  1473  1887  8648  2899  2039  2197  1053  3235   939
  1967  7253  2292  1597  1381  5842  2579  8471  2782  1042  3186  2028
   958  1591  2762  2779  2743  1149  1313  1190  3448 11328  1872  2058
  2136  1484   660  3394   609  1884  1620  2629   719  5096  1244  1842
  2576  1512 11054   518  2759  2670  4817  2679  3905  3386   343  4594
  3620  1721  3017   754  1950  2924  1659  7238  2764  4679  3092   448
   654  1238  1245  3114  2569  5152  1037  3573  1201  3622   960  1163
  1209  3077  3757  1418  3518  1934  8318   368  2122  2996  9034  1585
  1301  1323  3123  5493  1216  1207  1309  2360  6850  8588   759  4686
  2687   585  2255  1361  7127  1203   700  5507  3190  7119  3488  1113
  7966  1532  1503  2302   662  2273  2631  1311  3105  2319  3612  7763
  3049  1534  2032  6350  2864  1255  1333  2022  1552   626  8858   996
  1750  6999  1995  1331  2278  5003  3552  1928  2964  1546   683 12389
  4712  1553  1372  3979  6758  3234  5433   806  1082  2788  2930  1927
  2820   937  1056  3124  1388  2384  2133  2799  1289  1217  2246   385
```

```
  1965  1572  2718  1358   931  1442  4241  2775  3863  2329   918  1837
  3349  2828  4526  2671  2051  1300   741  3357  3632  1808 12204  9157
  3676  3441   640  3652  1530  3914  1858  2600  1979  2116  1437  4042
  3660  1444  1980  1355  1376 15653  1493  4370   750  1308  4623  1851
  1880  7980  4583  1386   947   684  7476  1922  2303  8086  2346  3973
   888 10222  4221  6361  1297   900  1050  1047  6314  3496  3609  4843
  4139  5742 10366  2080  2580  4530  5150  5595  1453  1538  2279  5103
  9857  6527  1347  2862  2753  3651   975  2896  4716  2284  1103   926
  1800  1905  1123  6331  1377  2503  2528  5324  6560  2969  1206  2118
   629  1198  2476  1138 14027  7596  1505  3148  6148  1337  1228   790
  2570   250  1316  1882  6416  6403  1987   760  2603  3380  3990 11560
  4380  6761  4280  2325  1048  3160  2483 14179  1797  2511  1274  5248
  3029   428   976   841  5771  1555  1285  1299  1271   691  5045  2124
  2214 12680  2463  1155  3108  2901  1655  2812  8065  3275  2223  1480
  1371  3535  3509  5711  3872  4933  1940   836  1941  2675  2751  6224
  5998  1188  6313  1221  2892  3062  2301  7511  1549  1795  7472  9271
   590   930  9283  1778   907   484  9629  3051  3931  7432  1338  1554
 15857  1345  1101  3016  2712   731  3780  1602  3966  4165  8335  6681
  2375 11816  5084  2327   886   601  2957  2611  5179  2993  1943  1559
  3422  3976  1249  2235  1471 10875   894  3343  3959  3577  5804  2169
  2439  2210  2221  2389  3331  7409   652  7678  1343   874  3590  1322
  3595  1422  6742  7814  9277  2181  1098  4057   795  2825 15672  6614
  7824  2442  1829  5800  8947  2606  1592  2186  4153  2625  3485 10477
  1278  1107  3763  3711  3594  3195  4454  4736  2991  2142  3161 18424
  2848 14896  2359  3345  1817 12749  1366  2002  6872   697  1049 10297
  1867  1747  1670  1224   522  1498   745  2063  6288  6842  3527   929
  1455  1845  8358  2859  3621  2145  4113 10974  1893  3656  4006  3069
  1740  2353  3556  2397   454  1715  2520  3568  7166  3939  1514  7393
  1193  7297  2831   753  2427  2538  8386  4844  2923  8229  1433  6289
  6579  3565  1569  1936  2390  1736  3857   804  4576]
Unique values in  Duration are : [ 6 48 12 42 24 36 30 15  9 10  7 60 18 45 1
1 27  8 54 20 14 33 21 16  4
 47 13 22 39 28  5 26 72 40]
Unique values in  Risk are : [1 0]
Unique values in  business are : [0 1]
Unique values in  domestic appliances are : [0 1]
Unique values in  education are : [0 1]
Unique values in  furniture/equipment are : [0 1]
Unique values in  radio/TV are : [1 0]
Unique values in  repairs are : [0 1]
Unique values in  vacation/others are : [0 1]
```

Plot heatmap

In [51]:
```
plt.figure(figsize = (12,12))
sns.heatmap(newdf.corr(),annot=True)
plt.show()
```



Select the Dependent(target) and Independent Variables:

In [52]:
```
X=newdf.drop(columns=['Risk'])
y=newdf['Risk']
```

Standardize the data

In [53]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X=sc.fit_transform(X)
```

In [54]:
```
pd.DataFrame(X, columns=newdf.drop(columns=['Risk']).columns.tolist()).head()
```

Out[54]:

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	busines
0	2.766456	0.670280	0.146949	-0.133710	1.833169	-1.254566	-0.745131	-1.236478	-0.32774
1	-1.191404	-1.491914	0.146949	-0.133710	-0.699707	-0.459026	0.949817	2.248194	-0.32774
2	1.183312	0.670280	-1.383771	-0.133710	-0.699707	1.132053	-0.416562	-0.738668	-0.32774
3	0.831502	0.670280	0.146949	-2.016956	-0.699707	-1.254566	1.634247	1.750384	-0.32774
4	1.535122	0.670280	0.146949	-2.016956	-0.699707	-1.254566	0.566664	0.256953	-0.32774

Split the data into training and testing set

In [55]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.2,random_state
```

Models:

Random Forest Classifier.

** Let's choose the best estimator and parameters :GridSearchCV**

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.model_selection import cross_val_score
```

In [57]:
```
model = RandomForestClassifier()
params = {'n_estimators':[16,32,50,100],'max_depth':[0.5,1,5,10],'random_state
          'n_jobs':[1,2]}
gsc = GridSearchCV(model, params, scoring = 'accuracy')
gsc.fit(X_train, y_train)
```

Out[57]:
```
GridSearchCV(estimator=RandomForestClassifier(),
             param_grid={'max_depth': [0.5, 1, 5, 10],
                         'n_estimators': [16, 32, 50, 100], 'n_jobs': [1, 2],
                         'random_state': [1, 10, 20, 42]},
             scoring='accuracy')
```

In [58]:
```
print("Best estimator is: ", gsc.best_estimator_)
print("Best parameters are: ", gsc.best_params_)
print("Best score is: ",gsc.best_score_)
```

```
Best estimator is:  RandomForestClassifier(max_depth=10, n_estimators=50, n_j
obs=1, random_state=1)
Best parameters are:  {'max_depth': 10, 'n_estimators': 50, 'n_jobs': 1, 'ran
dom_state': 1}
Best score is:  0.7474999999999999
```

In [59]:
```
model = gsc.best_estimator_
model.fit(X_train,y_train)
```

Out[59]:
```
RandomForestClassifier(max_depth=10, n_estimators=50, n_jobs=1, random_state=
1)
```

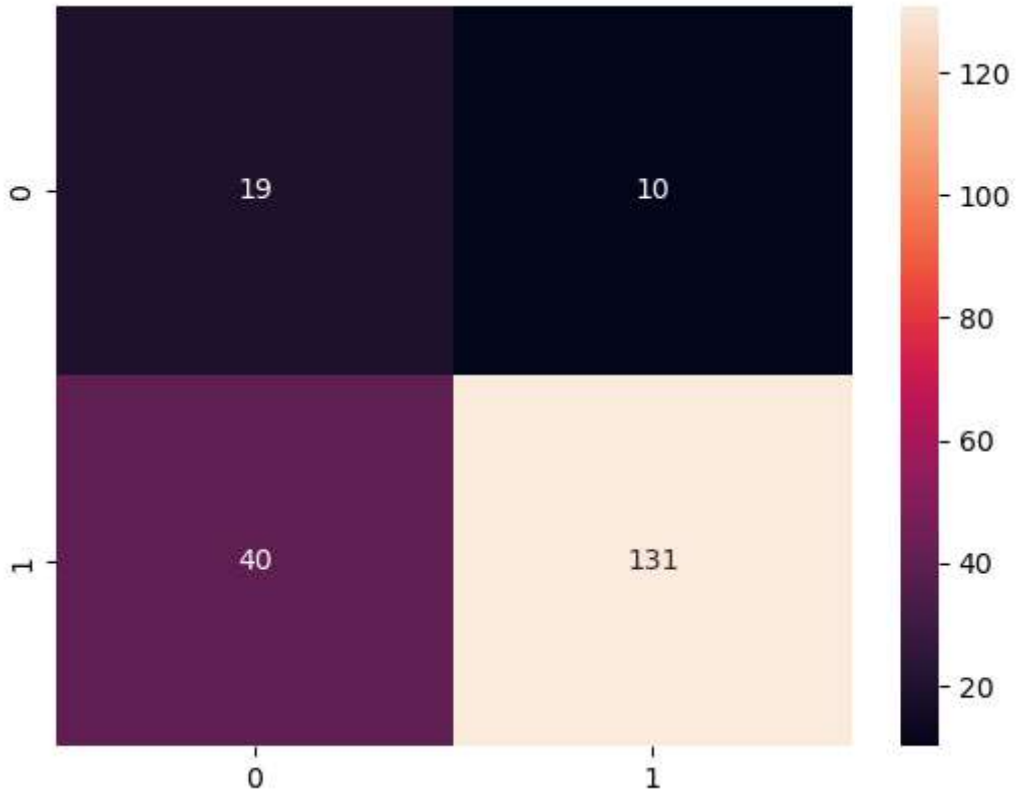
Check the accuracy score, Confusion metrics, Classification report and Cross val score

In [60]:
```
prediction = model.predict(X_test)
print("Accuracy score is" ,metrics.accuracy_score(prediction, y_test))
print("Confusion metrics:\n" ,metrics.confusion_matrix(prediction,y_test))
print("Classification report:\n", metrics.classification_report(prediction,y_t
rfscore = cross_val_score(model, X_train,y_train, scoring= "accuracy").mean()
print("Cross val score: ", rfscore)
```

```
Accuracy score is 0.75
Confusion metrics:
 [[ 19  10]
 [ 40 131]]
Classification report:
              precision    recall  f1-score   support

           0       0.32      0.66      0.43        29
           1       0.93      0.77      0.84       171

    accuracy                           0.75       200
   macro avg       0.63      0.71      0.64       200
weighted avg       0.84      0.75      0.78       200

Cross val score:  0.7474999999999999
```

Visualize the confusion matrix

In [61]:
```
sns.heatmap(metrics.confusion_matrix(prediction,y_test),annot=True, fmt='d')
plt.show()
```



SupportVectorClassifier:

In [62]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.model_selection import cross_val_score
```

In [63]:
```
model=SVC()
param_grid={'C': [0.75, 0.85, 0.95, 1], 'degree': [3, 4, 5],
                          'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}
gsc=GridSearchCV(model,param_grid,scoring="accuracy")
gsc.fit(X_train,y_train)
```

Out[63]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.75, 0.85, 0.95, 1], 'degree': [3, 4, 5],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']},
             scoring='accuracy')
```

In [66]:
```
print("Best estimator is: ", gsc.best_estimator_)
print("Best parameters are: ", gsc.best_params_)
print("Best score is: ",gsc.best_score_)
```

```
Best estimator is:  SVC(C=0.95)
Best parameters are:  {'C': 0.95, 'degree': 3, 'kernel': 'rbf'}
Best score is:  0.7375
```

In [67]:
```
model=gsc.best_estimator_
model.fit(X_train,y_train)
prediction = model.predict(X_test)
print("Accuracy score is" ,metrics.accuracy_score(prediction, y_test))
print("Confusion metrics:\n" ,metrics.confusion_matrix(prediction,y_test))
print("Classification report:\n", metrics.classification_report(prediction,y_t
svmscore = cross_val_score(model, X_train,y_train, scoring= "accuracy").mean()
print("Cross val score: ", svmscore)
```

```
Accuracy score is 0.76
Confusion metrics:
 [[ 19   8]
 [ 40 133]]
Classification report:
               precision    recall  f1-score   support

           0       0.32      0.70      0.44        27
           1       0.94      0.77      0.85       173

    accuracy                           0.76       200
   macro avg       0.63      0.74      0.64       200
weighted avg       0.86      0.76      0.79       200

Cross val score:  0.7375
```

LogisticRegression

In [68]:
```
from sklearn.linear_model import LogisticRegression

model =  LogisticRegression()
model.fit(X_train, y_train)
prediction = model.predict(X_test)
print("Accuracy score is" ,metrics.accuracy_score(prediction, y_test))
print("Confusion metrics:\n" ,metrics.confusion_matrix(prediction,y_test))
print("Classification report:\n", metrics.classification_report(prediction,y_t
logscore = cross_val_score(model, X_train,y_train, scoring= "accuracy").mean()
print("Cross val score: ", logscore)
```

```
Accuracy score is 0.75
Confusion metrics:
 [[ 21  12]
 [ 38 129]]
Classification report:
               precision    recall  f1-score   support

           0       0.36      0.64      0.46        33
           1       0.91      0.77      0.84       167

    accuracy                           0.75       200
   macro avg       0.64      0.70      0.65       200
weighted avg       0.82      0.75      0.77       200

Cross val score:  0.72125
```

AdaBoost: Classifier

In [69]:
```
from sklearn.ensemble import AdaBoostClassifier
model=AdaBoostClassifier()
model.fit(X_train,y_train)
prediction=model.predict(X_test)
print("Accuracy score is" ,metrics.accuracy_score(prediction, y_test))
print("Confusion metrics:\n" ,metrics.confusion_matrix(prediction,y_test))
print("Classification report:\n", metrics.classification_report(prediction,y_t
gradientscore = cross_val_score(model, X_train,y_train, scoring= "accuracy").m
print("Cross val score: ", gradientscore)
```

```
Accuracy score is 0.74
Confusion metrics:
 [[ 22  15]
 [ 37 126]]
Classification report:
              precision    recall  f1-score   support

           0       0.37      0.59      0.46        37
           1       0.89      0.77      0.83       163

    accuracy                           0.74       200
   macro avg       0.63      0.68      0.64       200
weighted avg       0.80      0.74      0.76       200

Cross val score:  0.72875
```

View scores of the different models

In [70]:
```
models = pd.DataFrame({'Models':['Random Forest Classifier','Logistic Regressi
                                 'Gradient Boost Classifier', 'Support Vector
                       'Score':[rfscore,logscore,gradientscore,svmscore]})
models.sort_values(by='Score', ascending = False)
```

Out[70]:

	Models	Score
0	Random Forest Classifier	0.74750
3	Support Vector Classifier	0.73750
2	Gradient Boost Classifier	0.72875
1	Logistic Regression	0.72125

ANALYZING THE RESULTS

So now we have to decide which one is the best model, and we have two types of wrong values:

- False Positive, means they won't pay the loan(Risk:Yes), but the model thinks they will.
- False Negative, means they will pay the loan(Risk:No), but the model said they won't.

In my opinion:

- Length of the dataset isn't enough, we need more data for better accuracy.

PCA to Speed up Machine Learning Algorithms (Logistic Regression)

Step 0: Import and use PCA. After PCA you will apply a machine learning algorithm of your choice to the transformed data

In [71]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=5)
pca.fit(X_train)
```

Out[71]:
```
PCA(n_components=5)
```

In [72]:
```
pca.n_components_
```

Out[72]:
```
5
```

In [73]:
```

train_X = pca.transform(X_train)
test_X = pca.transform(X_test)
```

Step 1: Import the model you want to use

In sklearn, all machine learning models are implemented as Python classes

In [74]:
```
from sklearn.linear_model import LogisticRegression
```

Step 2: Make an instance of the Model

In [75]:
```

logisticRegr = LogisticRegression(solver = 'lbfgs')
```

In [76]:
```
logisticRegr.fit(train_X,y_train)
```

Out[76]:
```
LogisticRegression()
```

In [77]:
```
logR_pred=logisticRegr.predict(test_X)
```

In [78]:
```
logisticRegr.score(test_X,y_test)
```

Out[78]:
```
0.71
```

In [79]:
```
from sklearn import metrics
metrics.confusion_matrix(logR_pred,y_test)
```

Out[79]:
```
array([[ 13,  12],
       [ 46, 129]], dtype=int64)
```

In [80]:
```
from sklearn.model_selection import cross_val_score
logR_cross_val_score = cross_val_score(logisticRegr,train_X,y_train, cv = 10).
logR_cross_val_score
```

Out[80]:
```
0.7125
```

Model Deployment With Tkinter

Import Tkinter Library

In [81]:
```
from tkinter import *
```

Define a function that will return the output of our prediction in an Entry box in tkinter window

In [82]:
```
def getPrediction():

    age1 = int(age.get())
    gender1 = gender.get()
    housing1 = housing.get()
    job1 = job.get()
    saving1 = savings.get()
    checking1 = checking.get()
    credit1 = int(credit.get())
    duration1 = int(duration.get())
    purpose1 = purpose.get()


    if gender1 == 'Male':
        gender1 = 0
    else:
        gender1 = 1

    if housing1 == 'Free':
        housing1 = 0
    elif housing1 == 'Own':
        housing1 = 1
    else:
        housing1 = 2


    if job1 == "Unskilled and Resident":
        job1 = 1
    elif job1 == "Skilled":
        job1 = 2
    elif job1 == "Highly Skilled":
        job1 = 3
    else:
        job1 = 0


    if saving1 == "Unknown":
        saving1 = 0
    elif saving1 == "Little":
        saving1 = 1
    elif saving1 == "Moderate":
        saving1 = 2
    elif saving1 == "Quite Rich":
        saving1 = 3
    else:
        saving1 = 4

    if checking1 == "Unknown":
        checking1 = 0
    elif checking1 == "Little":
        checking1 = 1
    elif checking1 == "Moderate":
        checking1 = 2
    else:
        checking1 = 3
```

```
    if purpose1 == "Radio/TV":
        business = 0
        domestic = 0
        education = 0
        furniture = 0
        radio = 1
        repairs = 0
        vacation = 0
    elif purpose1 == "Education":
        business = 0
        domestic = 0
        education = 1
        furniture = 0
        radio = 0
        repairs = 0
        vacation = 0
    elif purpose1 == "Furniture":
        business = 0
        domestic = 0
        education = 0
        furniture = 1
        radio = 0
        repairs = 0
        vacation = 0
    elif purpose1 == "Car":
        business = 0
        domestic = 0
        education = 0
        furniture = 0
        radio = 0
        repairs = 0
        vacation = 0
    elif purpose1 == "Business":
        business = 1
        domestic = 0
        education = 0
        furniture = 0
        radio = 0
        repairs = 0
        vacation = 0
    elif purpose1 == "Domestic Appliance":
        business = 0
        domestic = 1
        education = 0
        furniture = 0
        radio = 0
        repairs = 0
        vacation = 0
    elif purpose1 == "Repairs":
        business = 0
        domestic = 0
        education = 0
        furniture = 0
        radio = 0
        repairs = 1
        vacation = 0
    else:
```

```
        business = 0
        domestic = 0
        education = 0
        furniture = 0
        radio = 0
        repairs = 0
        vacation = 1


    x = [[age1, gender1, job1, housing1, saving1, checking1, credit1, duration

    x = sc.transform(x)

    x = pca.transform(x)

    result = logisticRegr.predict(x)

    resultbox.delete(0, END)

    if result[0] == 1:
        resultbox.insert(0, 'Good')
    elif result[0] == 0:
        resultbox.insert(0, 'Bad')
    else:
        resultbox.insert(0, 'Error')
```

Create Tkinter window and all necessary input and output boxes

In [ ]:

```
window = Tk()


window.geometry("450x550")

window.title('German Credit Risk Analysis')

label = Label(window , text = 'Enter the Details here:', font=('calibre',10, '

agelabel = Label(window, text = 'Age').pack()
age = Entry(window)
age.pack()

gender = StringVar()
genderlabel = Label(window, text = "Gender").pack()
gender_options = ["Male", "Female"]
gender.set(gender_options[0])
gender_drop = OptionMenu(window, gender, *gender_options).pack()

housing = StringVar()
housinglabel = Label(window, text = "Housing").pack()
housing_options = ["Free", "Own", "Rent"]
housing.set(housing_options[0])
housing_drop = OptionMenu(window, housing, *housing_options).pack()

job = StringVar()
joblabel = Label(window, text = "Job").pack()
job_options = ['Unskilled and Resident', 'Skilled', 'Highly Skilled', 'Unskill
job.set(job_options[0])
jobdrop = OptionMenu(window, job, *job_options).pack()

savings = StringVar()
savingslabel = Label(window, text = "Savings Account").pack()
savings_options = ['Unknown', 'Little','Moderate', 'Quite Rich','Rich']
savings.set(savings_options[0])
savingsdrop = OptionMenu(window , savings , *savings_options ).pack()

checking = StringVar()
checkinglabel = Label(window, text = "Checking Account").pack()
checking_options = ['Unknown', 'Little', 'Moderate','Rich']
checking.set(checking_options[0])
checkingdrop = OptionMenu(window , checking , *checking_options ).pack()

creditlabel = Label(window, text = 'Credit Amount').pack()
credit = Entry(window)
credit.pack()

durationlabel = Label(window, text = "Duration in Months").pack()
duration = Entry(window)
duration.pack()

purpose = StringVar()
purposelabel = Label(window, text = "Purpose of Credit").pack()
purpose_options = ["Radio/TV", "Education", "Furniture", "Car", "Business","Do
purpose.set(purpose_options[0])
purposedrop = OptionMenu(window, purpose, *purpose_options).pack()
```

```
button = Button(window, text = 'Get Prediction', command = getPrediction).pack


resultlabel = Label(window, text = "Credit Risk").pack()
resultbox = Entry(window)
resultbox.pack()

window.mainloop()
```

```
C:\Users\DELL\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but StandardScaler was fitted with feature
names
  warnings.warn(
```

In [ ]: