Importing Data

In [1]:
```
import numpy as np
import pandas as pd
data1=pd.read_csv('/Users/bbkpa/Downloads/train (3).csv')
data2=pd.read_csv('/Users/bbkpa/Downloads/test.csv')
data1
```

Out[1]:

	customer_id	name	age	gender	owns_car	owns_house	no_of_children	net_yearly_income	no_of_days_employed	oc
0	CST_115179	ita Bose	46	F	N	Y	0.0	107934.04	612.0	
1	CST_121920	Alper Jonathan	29	M	N	Y	0.0	109862.62	2771.0	
2	CST_109330	Umesh Desai	37	M	N	Y	0.0	230153.17	204.0	
3	CST_128288	Rie	39	F	N	Y	0.0	122325.82	11941.0	
4	CST_151355	McCool	46	M	Y	Y	0.0	387286.00	1459.0	
...	...	...	...	...	...	...	...	...	...	
45523	CST_130421	Doris	55	F	N	N	2.0	96207.57	117.0	
45524	CST_136670	Luciana	31	F	N	Y	0.0	383476.74	966.0	
45525	CST_145435	Jessica	27	F	N	Y	0.0	260052.18	1420.0	
45526	CST_130913	Tessa	32	M	Y	N	0.0	157363.04	2457.0	
45527	CST_160078	Gopinath	38	M	N	Y	1.0	316896.28	1210.0	

45528 rows × 19 columns

In [2]:
```
data2
```

Out[2]:

	customer_id	name	age	gender	owns_car	owns_house	no_of_children	net_yearly_income	no_of_days_employed	oc
0	CST_142525	Siva	52	F	Y	N	0.0	232640.53	998.0	
1	CST_129215	Scott	48	F	N	N	1.0	284396.79	1338.0	
2	CST_138443	Victoria	50	F	N	N	1.0	149419.28	1210.0	
3	CST_123812	John McCrank	30	F	N	N	1.0	160437.54	503.0	
4	CST_144450	Martinne	52	M	N	Y	0.0	233480.37	157.0	
...	...	...	...	...	...	...	...	...	...	
11378	CST_142412	Solarina	53	F	N	N	0.0	266824.38	3051.0	
11379	CST_107967	Jonathan Cable	33	F	NaN	N	0.0	124310.85	365248.0	
11380	CST_134002	Robin	27	M	Y	Y	1.0	364652.81	3431.0	
11381	CST_146856	Lauren	36	F	N	Y	0.0	128769.02	16320.0	
11382	CST_112001	Lynnley Browning	45	F	N	Y	0.0	158543.43	9443.0	

11383 rows × 18 columns

In [3]:
```
combined_data=pd.concat([data2,data1],axis=0)
```

we combined both datasets

In [5]:
```
data=combined_data.sample(frac=1,random_state=42).reset_index(drop=True)
```

In [6]:
```
data
```

Out[6]:

	customer_id	name	age	gender	owns_car	owns_house	no_of_children	net_yearly_income	no_of_days_employed	oc
0	CST_151936	Shanley	39	F	N	Y	0.0	160503.35	1154.0	
1	CST_153713	Kambas	32	F	N	Y	1.0	310268.73	239.0	
2	CST_165771	Asokan	27	M	Y	N	0.0	264593.49	4014.0	
3	CST_118039	John McCrank	24	F	N	Y	0.0	170396.97	4189.0	
4	CST_144960	Deepa	44	F	N	Y	0.0	222185.59	8438.0	
...	...	...	...	...	...	...	...	...	...	
56906	CST_148445	Edwards	38	F	Y	Y	1.0	78132.67	3479.0	
56907	CST_108380	Kirstin Ridley	52	F	N	Y	0.0	220697.50	365249.0	
56908	CST_161047	Clark	24	F	N	N	0.0	87469.36	3979.0	
56909	CST_162696	Santa	35	F	N	Y	1.0	160382.15	9322.0	
56910	CST_148157	Zieminski	48	F	N	Y	0.0	210916.56	365245.0	

56911 rows × 19 columns

No of missing values corresponding to each column

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

Out[8]:
```
customer_id                  0
name                         0
age                          0
gender                       0
owns_car                   679
owns_house                   0
no_of_children             964
net_yearly_income            0
no_of_days_employed        568
occupation_type              0
total_family_members       114
migrant_worker             113
yearly_debt_payments       117
credit_limit                 0
credit_limit_used(%)         0
credit_score                11
prev_defaults                0
default_in_last_6months      0
credit_card_default      11383
dtype: int64
```

Dropping unnecessary columns

In [10]:
```
data=data.drop(columns=['customer_id','name'])
```

Dropping all values which not present in our output column

In [11]:
```
data=data.dropna(subset=['credit_card_default'])
```

In [12]:
```
data
```

Out[12]:

	age	gender	owns_car	owns_house	no_of_children	net_yearly_income	no_of_days_employed	occupation_type	total_fam
0	39	F	N	Y	0.0	160503.35	1154.0	Cooking staff	
1	32	F	N	Y	1.0	310268.73	239.0	Core staff	
2	27	M	Y	N	0.0	264593.49	4014.0	Laborers	
3	24	F	N	Y	0.0	170396.97	4189.0	Medicine staff	
4	44	F	N	Y	0.0	222185.59	8438.0	Core staff	
...	...	...	...	...	...	...	...	...	
56905	29	M	Y	N	0.0	174277.82	815.0	Managers	
56906	38	F	Y	Y	1.0	78132.67	3479.0	Unknown	
56907	52	F	N	Y	0.0	220697.50	365249.0	Unknown	
56909	35	F	N	Y	1.0	160382.15	9322.0	Unknown	
56910	48	F	N	Y	0.0	210916.56	365245.0	Unknown	

45528 rows × 17 columns

Finding Categorical Columns

In [13]:
```
categorical_column=data.select_dtypes(include=['object','category'])
categorical_column
```

Out[13]:

	gender	owns_car	owns_house	occupation_type
0	F	N	Y	Cooking staff
1	F	N	Y	Core staff
2	M	Y	N	Laborers
3	F	N	Y	Medicine staff
4	F	N	Y	Core staff
...	...	...	...	...
56905	M	Y	N	Managers
56906	F	Y	Y	Unknown
56907	F	N	Y	Unknown
56909	F	N	Y	Unknown
56910	F	N	Y	Unknown

45528 rows × 4 columns

In [14]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

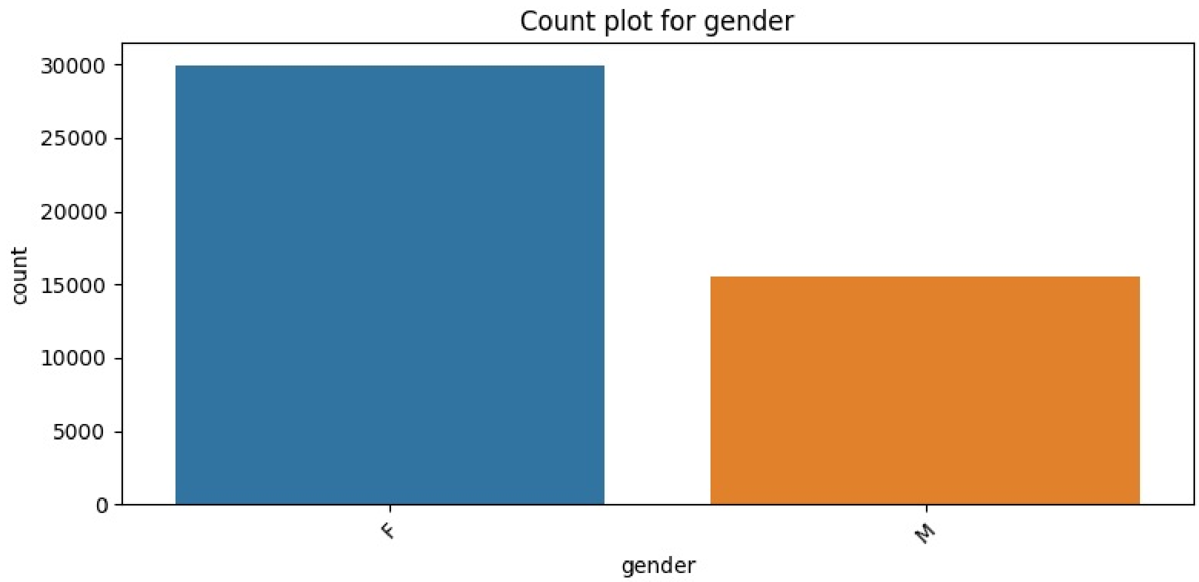
Exploratory Data Analysis

In [15]:
```
# Countplot of each categorical column in the dataset
```

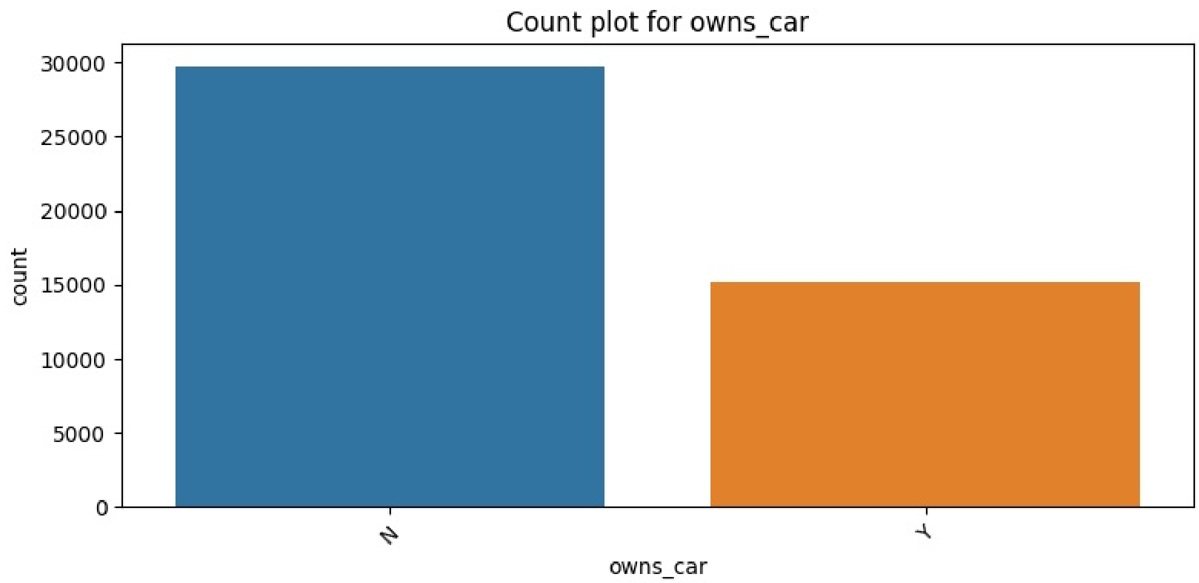
In [16]:
```
data=data[data['gender']!='XNA']
```

In [17]:
```
for col in data.select_dtypes(include=['object', 'category']).columns:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=data, x=col)
    print(data[col].value_counts())
    plt.title(f'Count plot for {col}')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

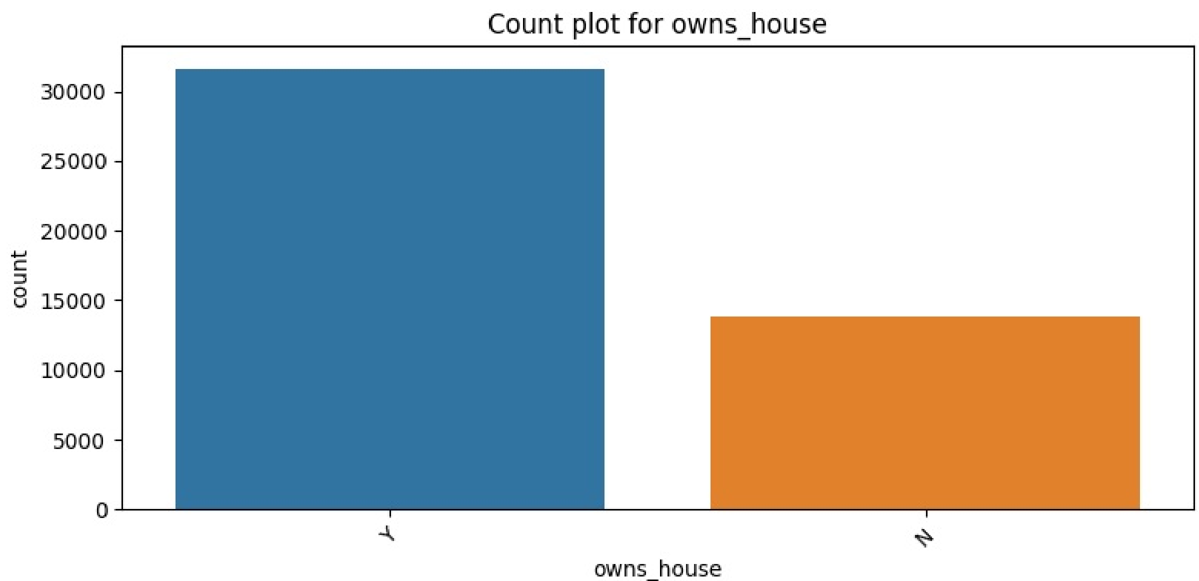
```
gender
F    29957
M    15570
Name: count, dtype: int64
```



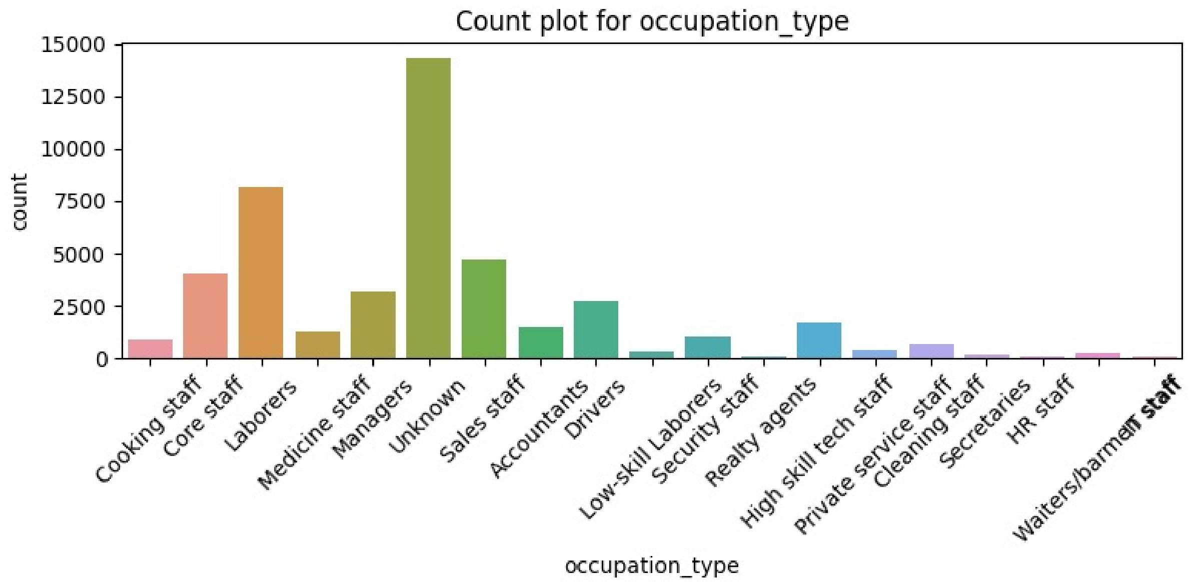

```
owns_car
N    29742
Y    15238
Name: count, dtype: int64
```




```
owns_house
Y    31641
N    13886
Name: count, dtype: int64
```



```
occupation_type
Unknown                  14299
Laborers                  8134
Sales staff               4725
Core staff                4062
Managers                  3168
Drivers                   2747
High skill tech staff     1682
Accountants               1474
Medicine staff            1275
Security staff            1025
Cooking staff              902
Cleaning staff             665
Private service staff      387
Low-skill Laborers         335
Waiters/barmen staff       203
Secretaries                199
Realty agents              101
HR staff                    78
IT staff                    66
Name: count, dtype: int64
```



In [ ]:
```
# All numerical columns of the dataset
```

In [18]:
```
data3=data[['age','net_yearly_income','no_of_days_employed','yearly_debt_payments','credit_limit','credit_limit_
```

In [19]:
```
data3
```

Out[19]:

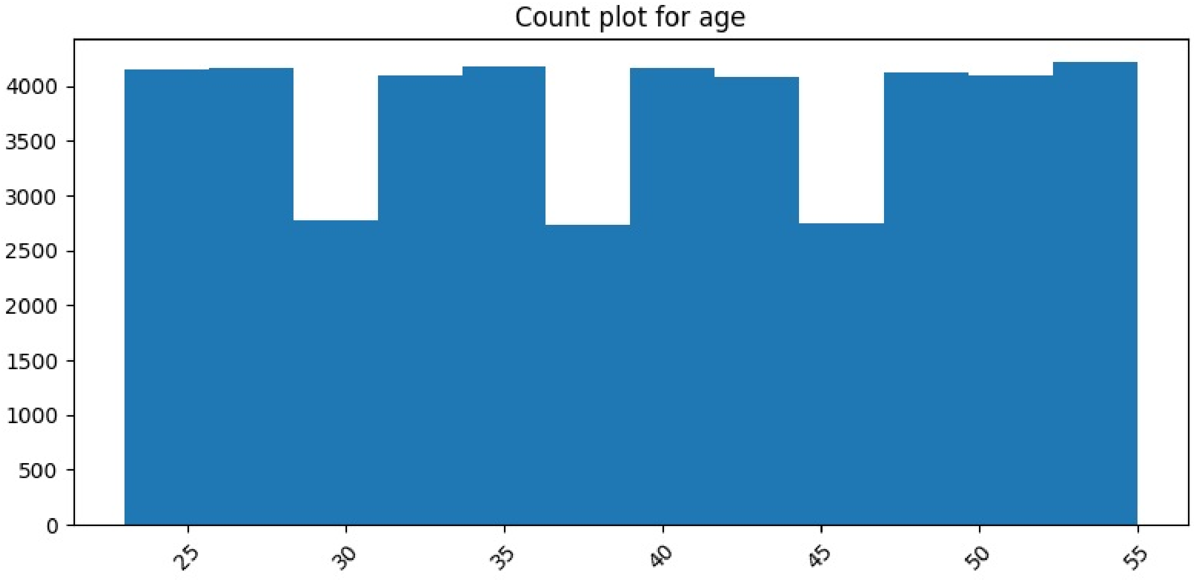
	age	net_yearly_income	no_of_days_employed	yearly_debt_payments	credit_limit	credit_limit_used(%)	credit_score
0	39	160503.35	1154.0	39985.72	31473.79	44	941.0
1	32	310268.73	239.0	79226.00	102848.89	36	781.0
2	27	264593.49	4014.0	25533.87	51100.55	59	929.0
3	24	170396.97	4189.0	19420.45	33075.39	80	840.0
4	44	222185.59	8438.0	35546.25	42287.74	87	840.0
...	...	...	...	...	...	...	...
56905	29	174277.82	815.0	29187.39	32810.34	54	830.0
56906	38	78132.67	3479.0	27837.35	9734.91	86	758.0
56907	52	220697.50	365249.0	31334.76	39506.32	53	728.0
56909	35	160382.15	9322.0	6578.35	36042.59	0	745.0
56910	48	210916.56	365245.0	32292.10	40379.72	95	842.0

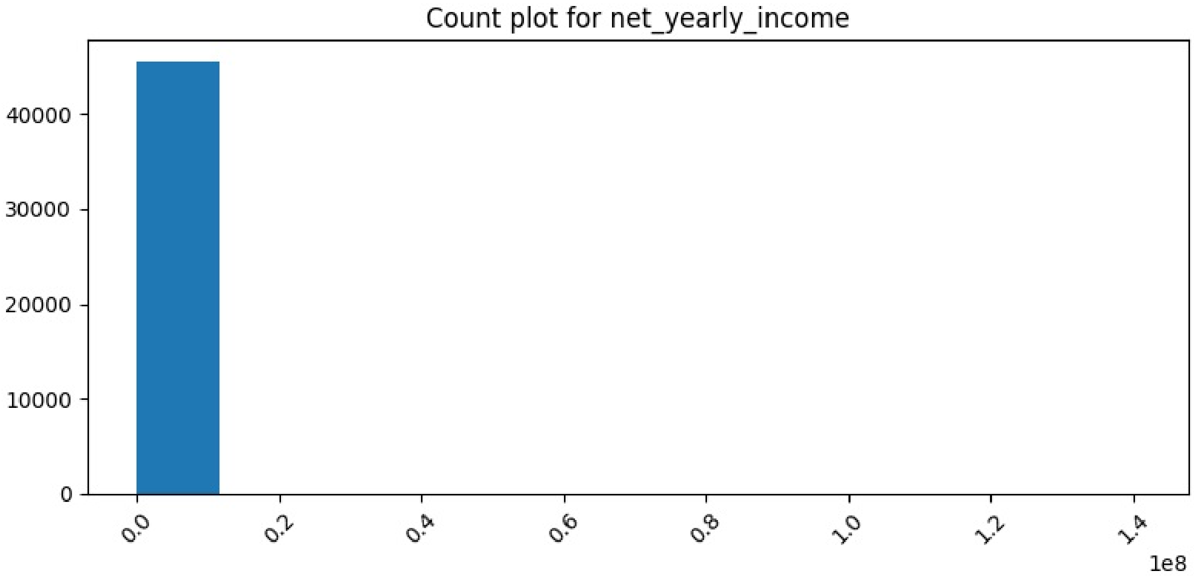
45527 rows × 7 columns

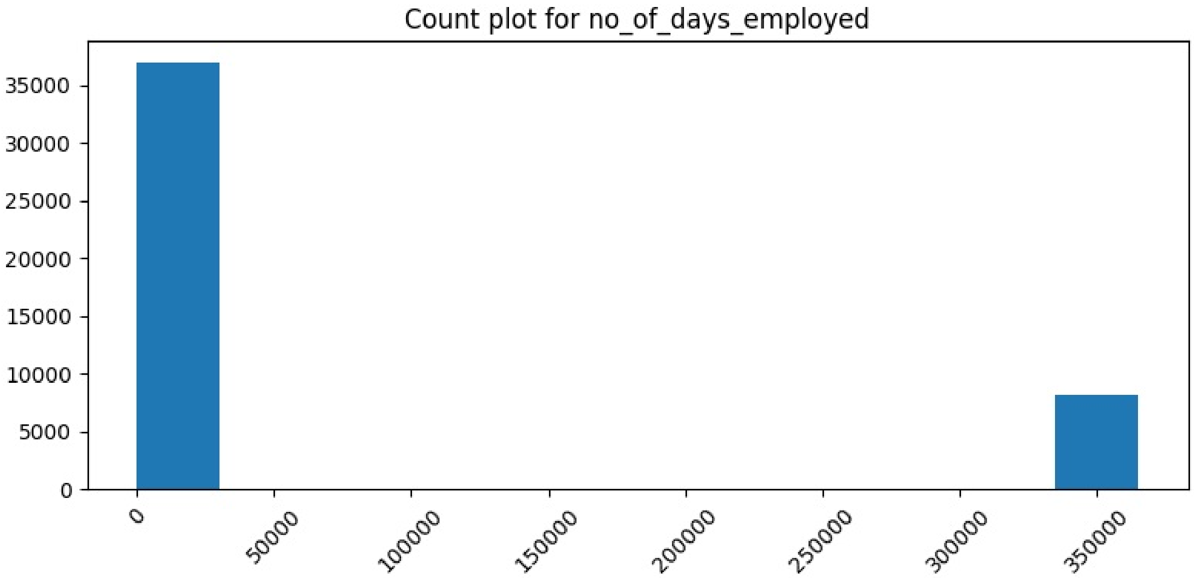
In [20]:
```
# Histogram of each numerical column
```

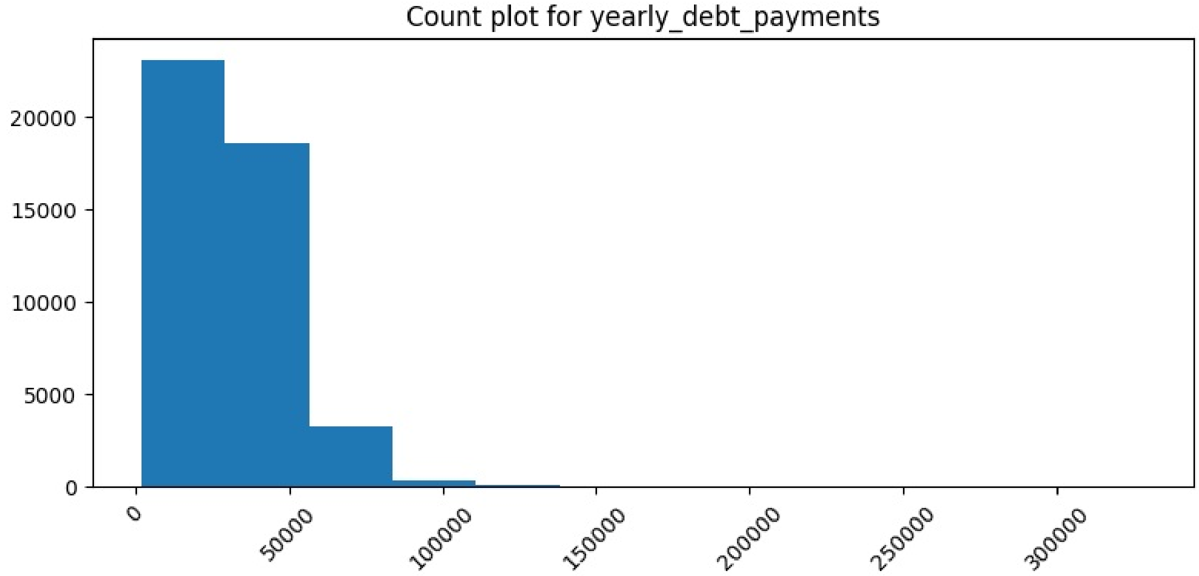
In [21]:
```
for col in data3:
    plt.figure(figsize=(8, 4))
    plt.hist(data3[col],bins=12)

    plt.title(f'Count plot for {col}')
```

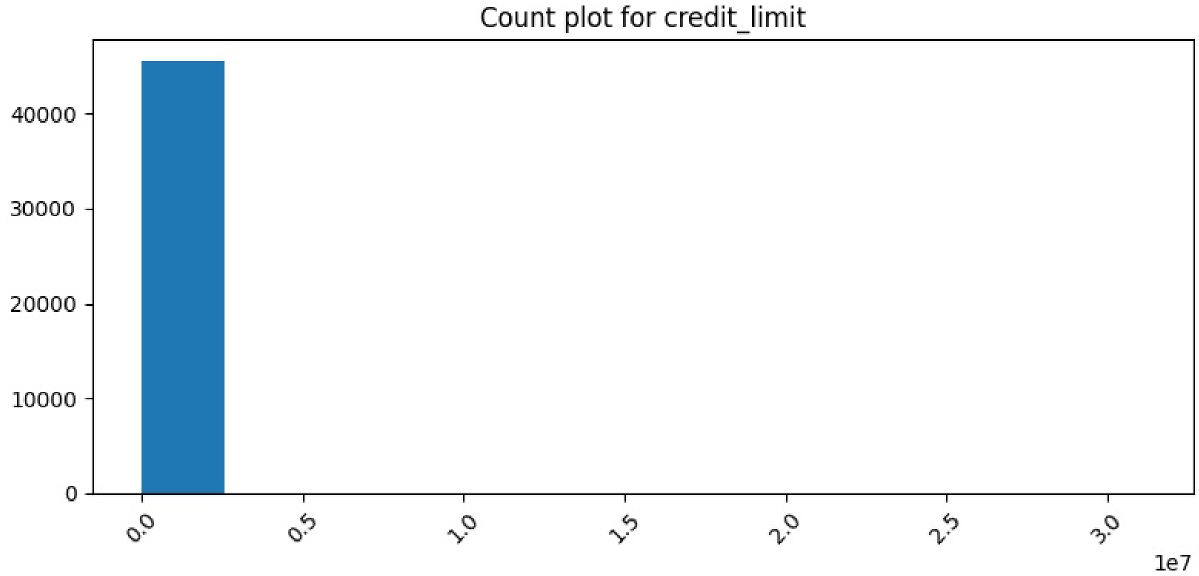
```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

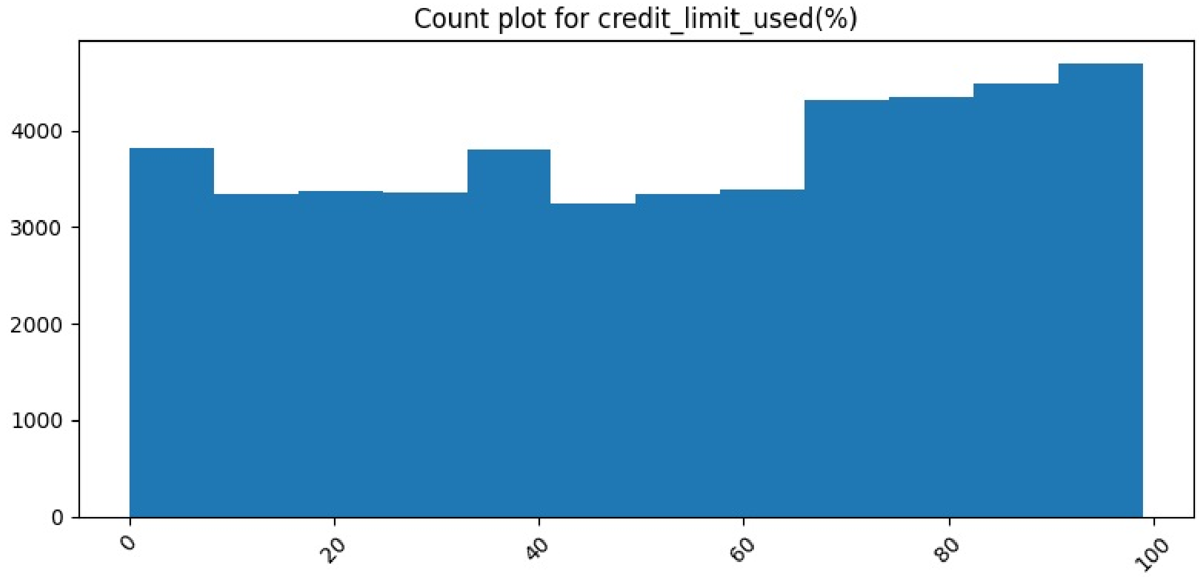


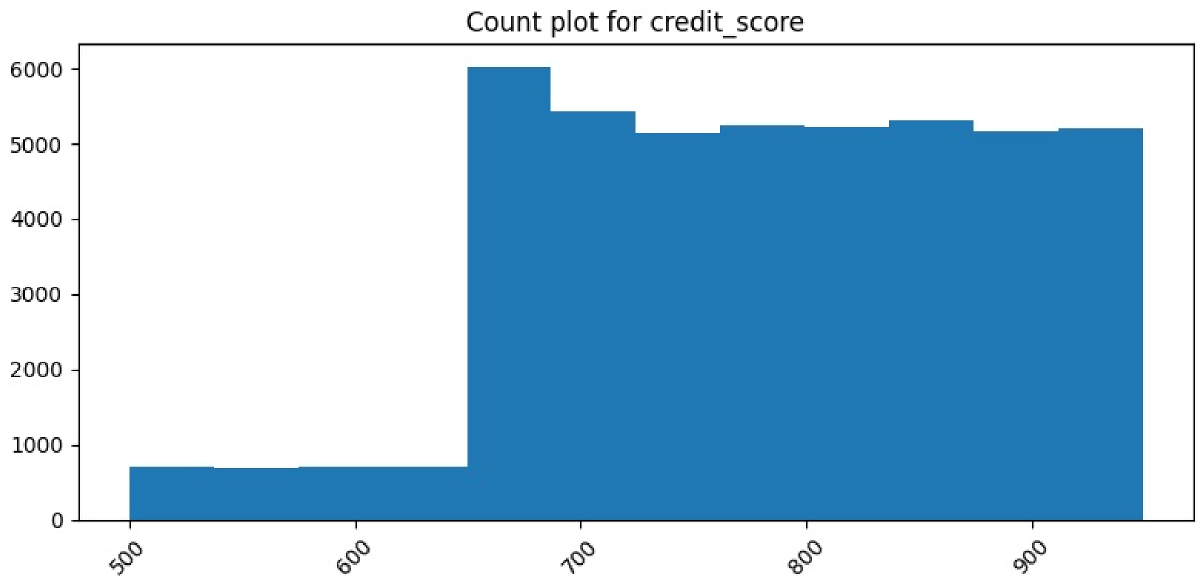





Count plot for yearly_debt_payments
0
5000
10000
15000
20000
0
50000
100000
150000
200000
250000
300000


Count plot for credit_limit
0
10000
20000
30000
40000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e7


Count plot for credit_limit_used(%)
0
1000
2000
3000
4000
0
20
40
60
80
100

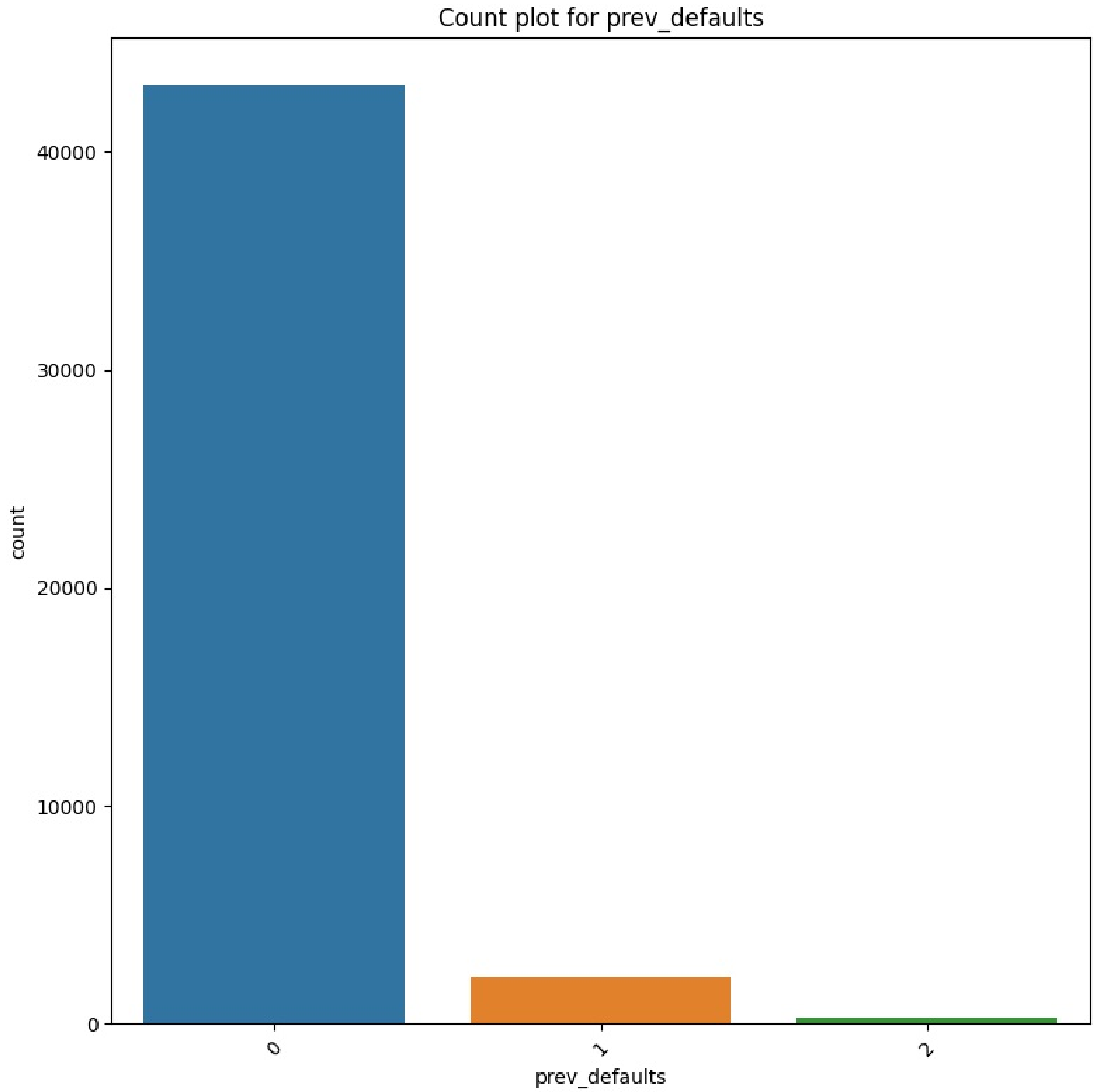


In [22]:
```
l=['prev_defaults','default_in_last_6months','credit_card_default','no_of_children','migrant_worker','total_fam:
```

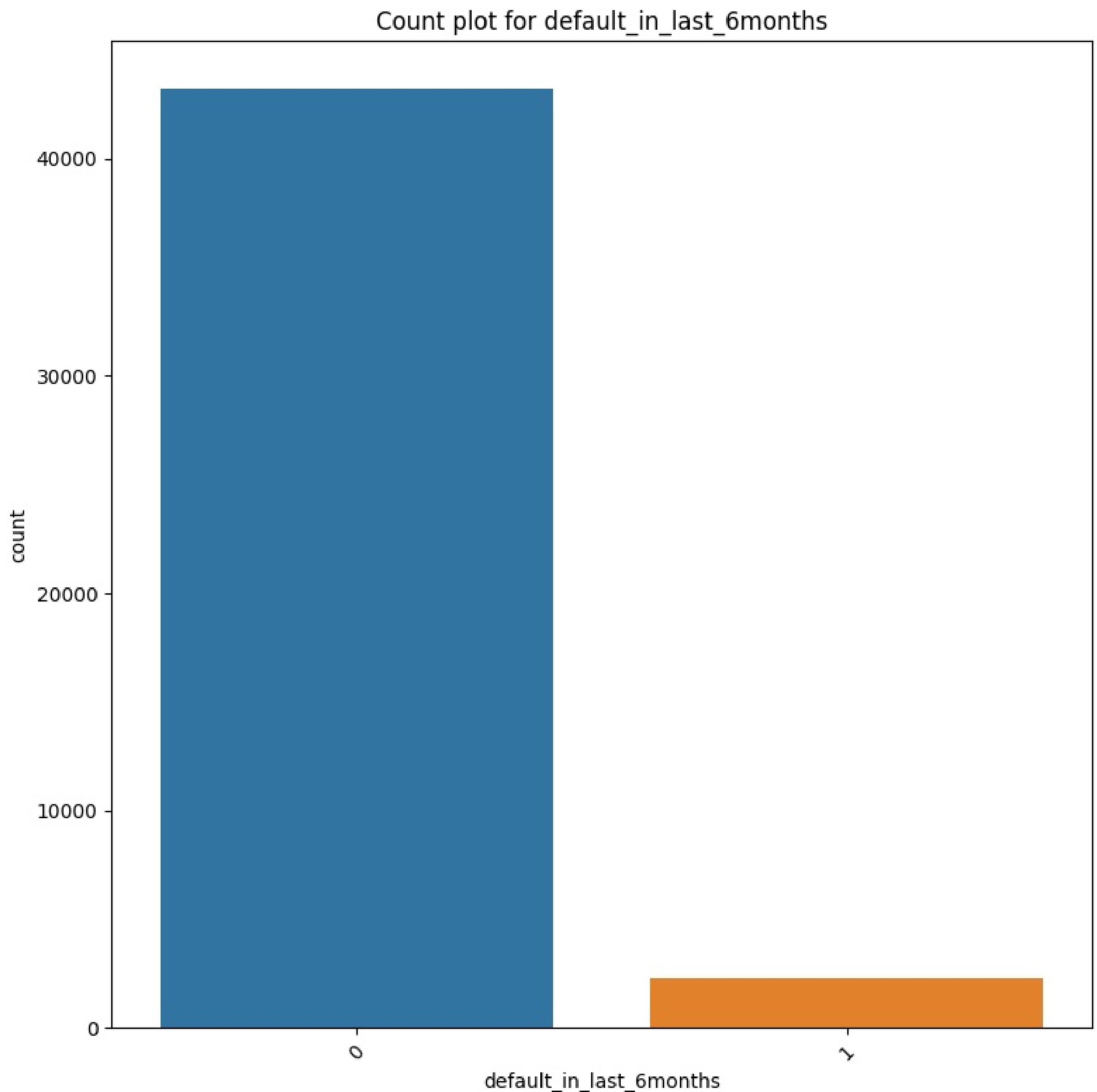
In [23]:
```
# Countplot of other categorical columns
```

In [24]:
```
for col in l:
    plt.figure(figsize=(8,8))
    sns.countplot(data=data,x=col)
    print(data[col].value_counts())
    plt.title(f'Count plot for {col}')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

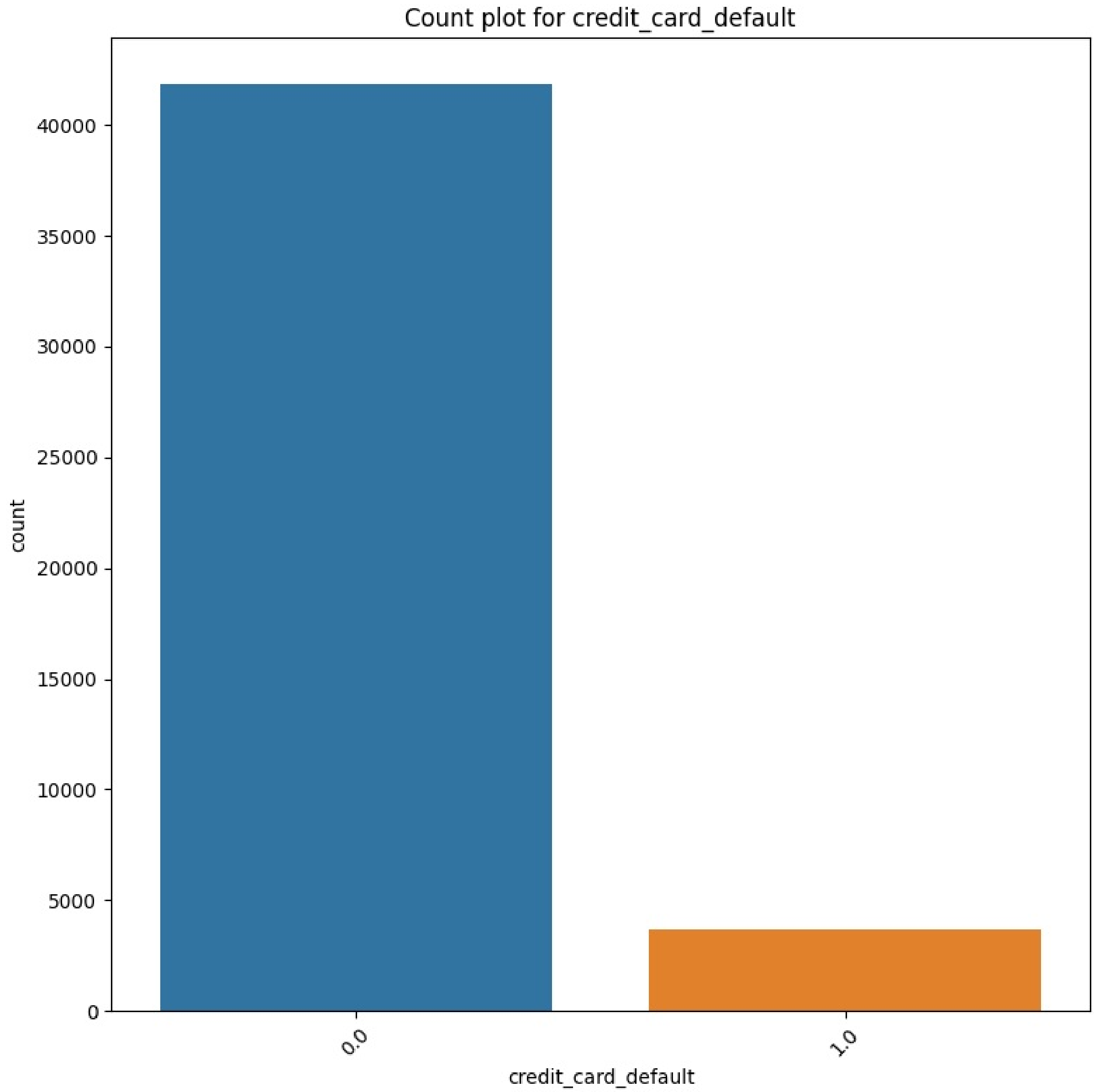
```
prev_defaults
0    43059
1     2172
2      296
Name: count, dtype: int64
```



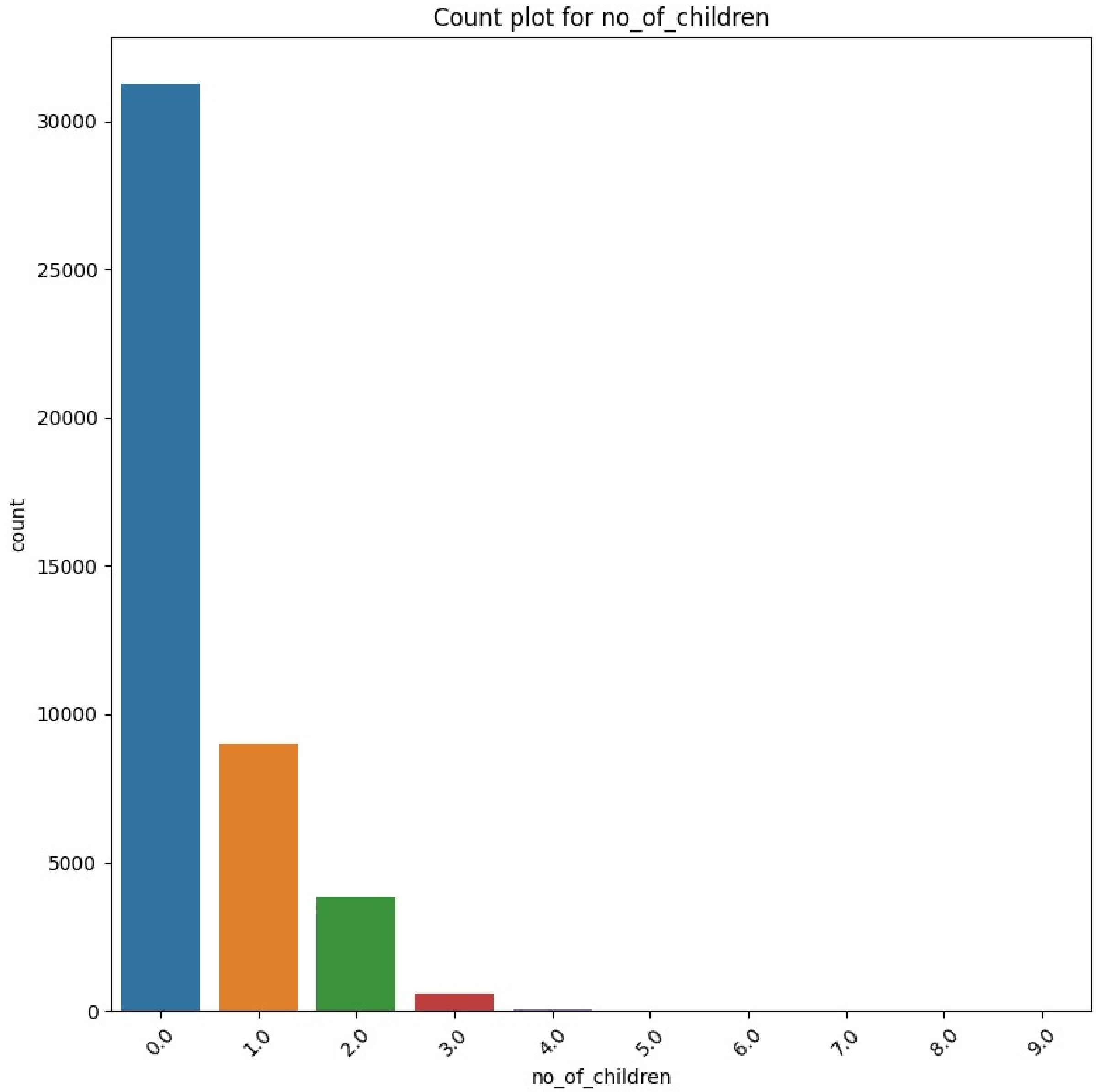

```
default_in_last_6months
0    43226
1     2301
Name: count, dtype: int64
```



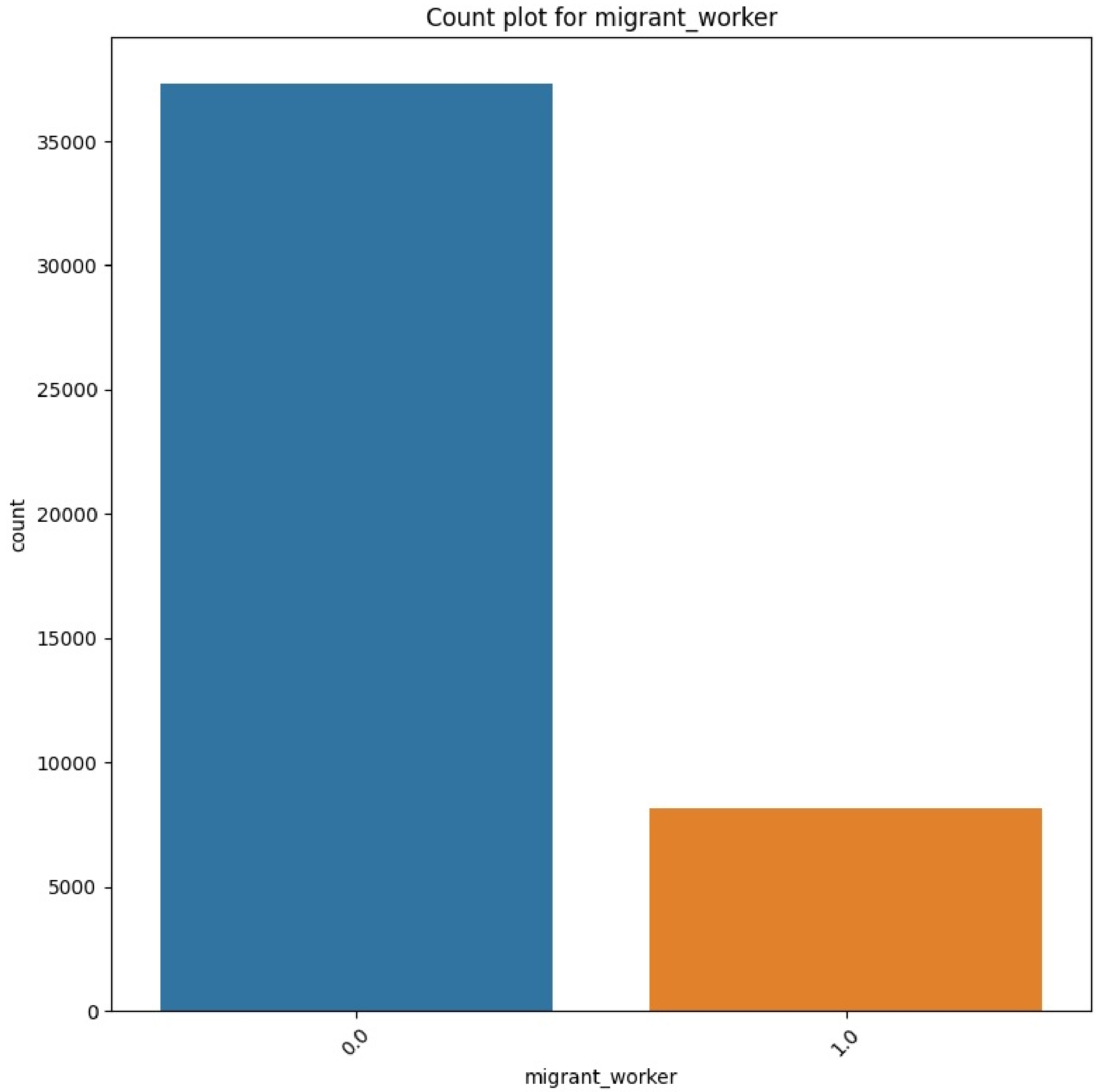

```
credit_card_default
0.0    41830
1.0     3697
Name: count, dtype: int64
```



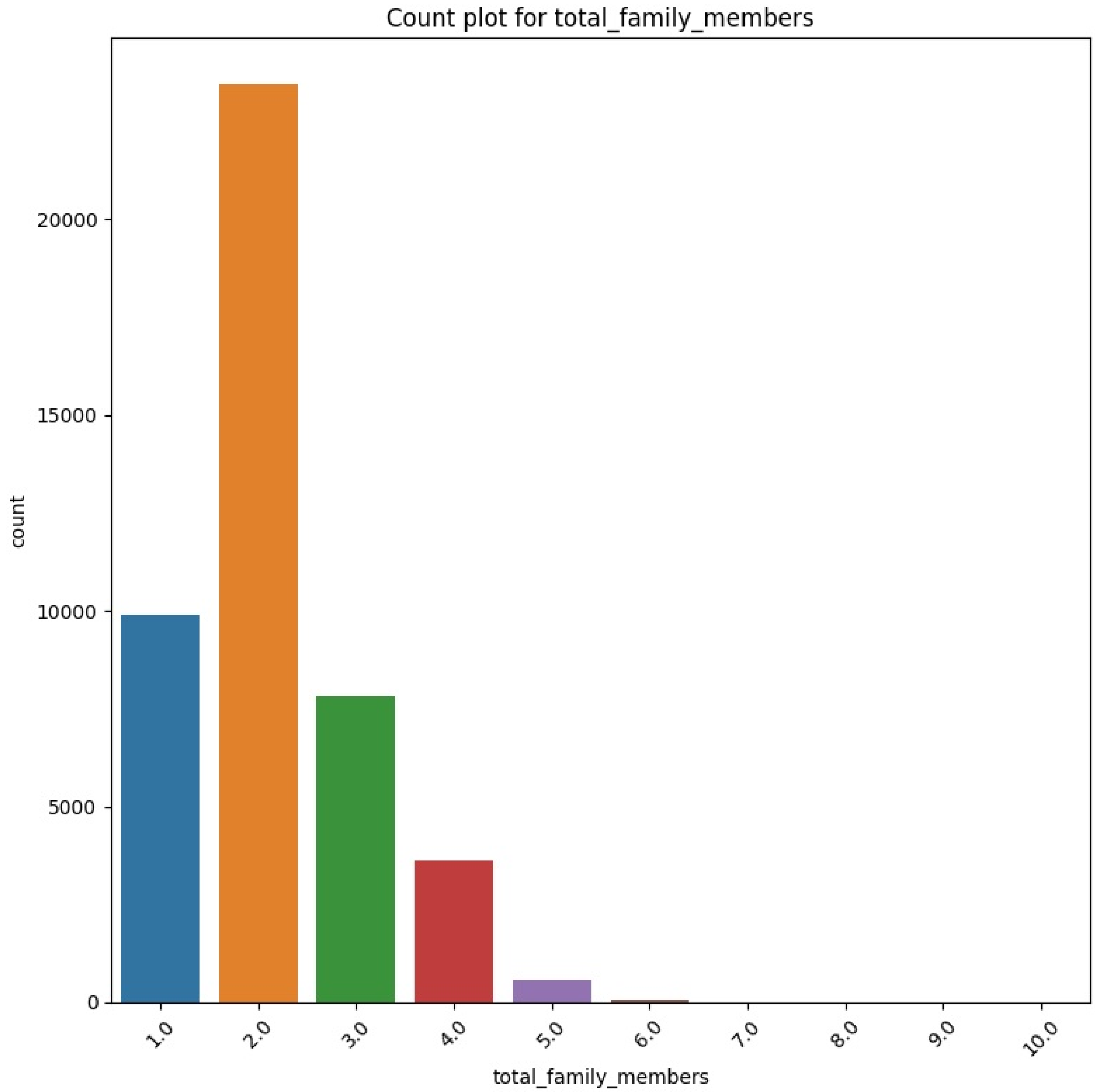

```
no_of_children
0.0    31241
1.0     8985
2.0     3861
3.0      584
4.0       60
5.0       13
6.0        6
8.0        1
7.0        1
9.0        1
Name: count, dtype: int64
```




```
migrant_worker
0.0    37301
1.0     8139
Name: count, dtype: int64
```




```
total_family_members
2.0     23455
1.0      9913
3.0      7812
4.0      3622
5.0       564
6.0        57
7.0        12
8.0         6
10.0        2
9.0         1
Name: count, dtype: int64
```



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

Out[25]:
```
no_of_days_employed
365246.0    684
365244.0    669
365240.0    641
365245.0    631
365241.0    628
           ...
12056.0       1
7531.0        1
11855.0       1
6021.0        1
9322.0        1
Name: count, Length: 7874, dtype: int64
```

In [26]:
```
# Total number of family members in a families
```

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

Out[27]:
```
total_family_members
2.0     23455
1.0      9913
3.0      7812
4.0      3622
5.0       564
6.0        57
7.0        12
8.0         6
10.0        2
9.0         1
Name: count, dtype: int64
```

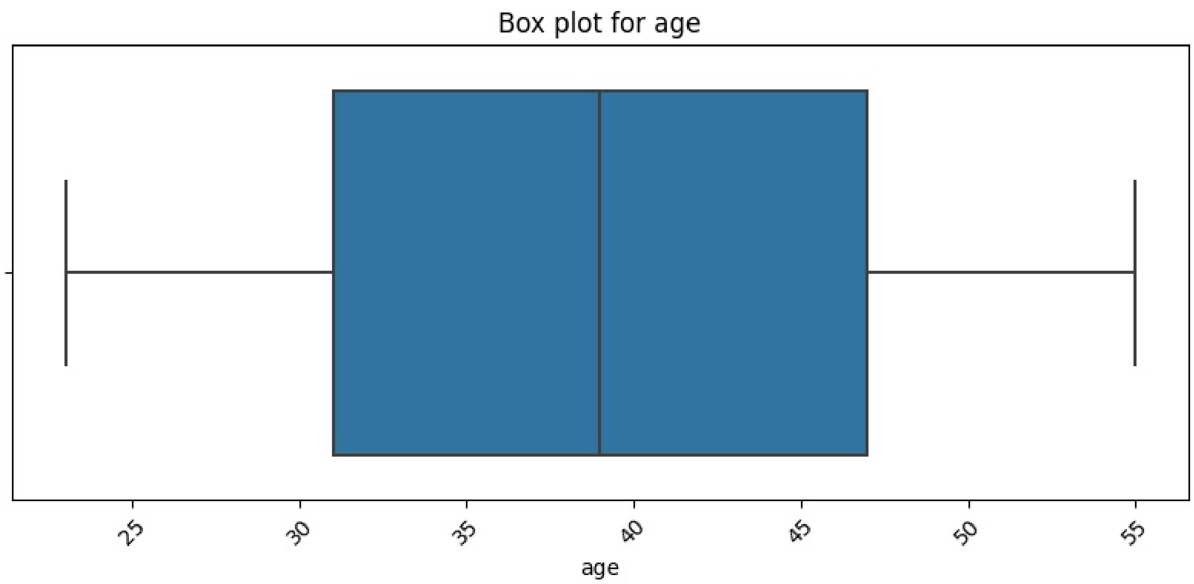
In [ ]:
```
# After dropping values from output column no of missing values in the dataset
```

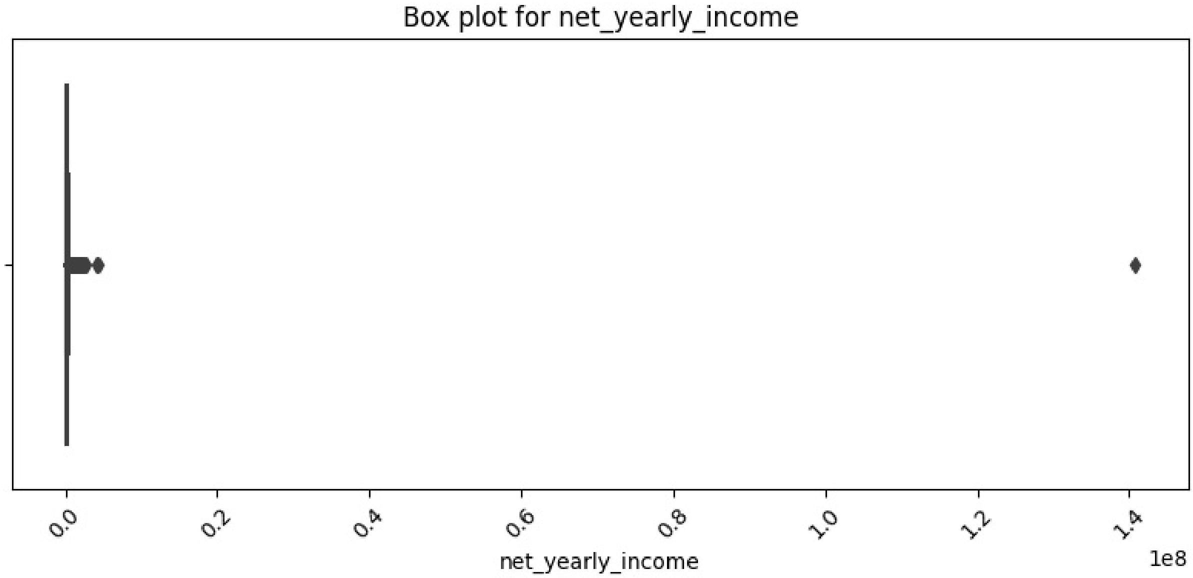
In [28]:
```
data.isnull().sum()
```

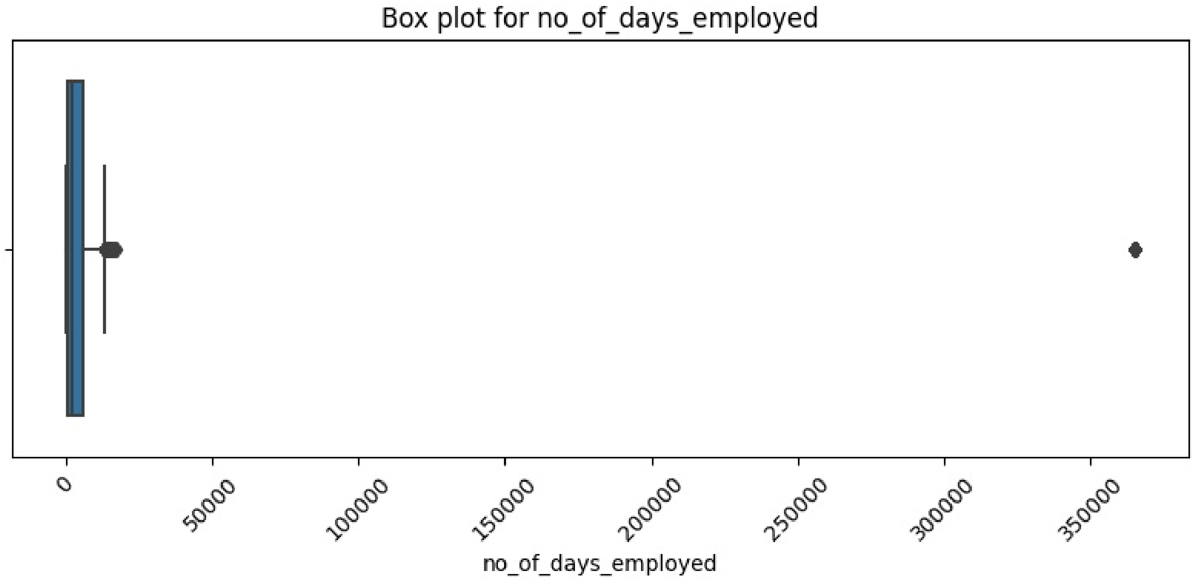
Out[28]:
```
age                          0
gender                       0
owns_car                   547
owns_house                   0
no_of_children             774
net_yearly_income            0
no_of_days_employed        463
occupation_type              0
total_family_members        83
migrant_worker              87
yearly_debt_payments        95
credit_limit                 0
credit_limit_used(%)         0
credit_score                 8
prev_defaults                0
default_in_last_6months      0
credit_card_default          0
dtype: int64
```

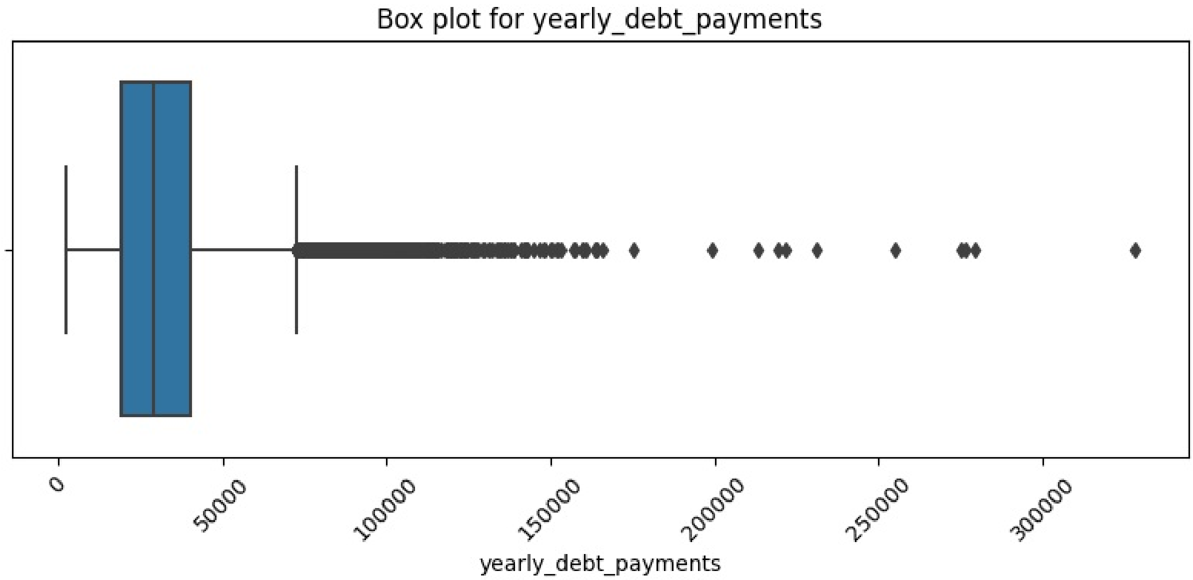
In [30]:
```
# To check number of outliers in our dataset
```

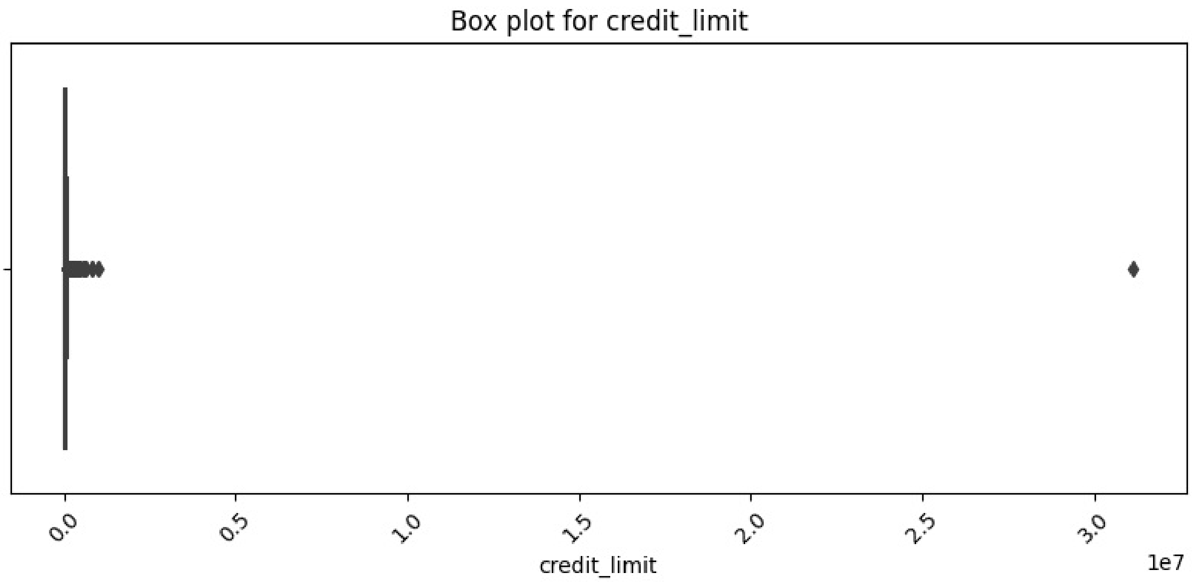
In [31]:
```
for col in data3:
     plt.figure(figsize=(8, 4))
     sns.boxplot(data=data3,x=col)

     plt.title(f'Box plot for {col}')
     plt.xticks(rotation=45)
     plt.tight_layout()
     plt.show()
```

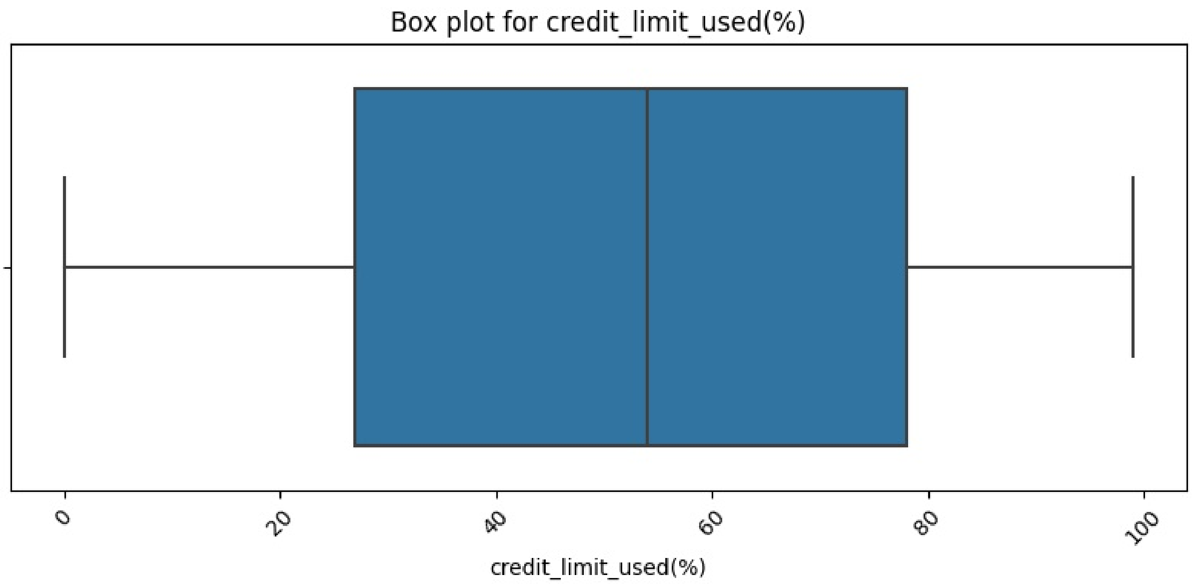


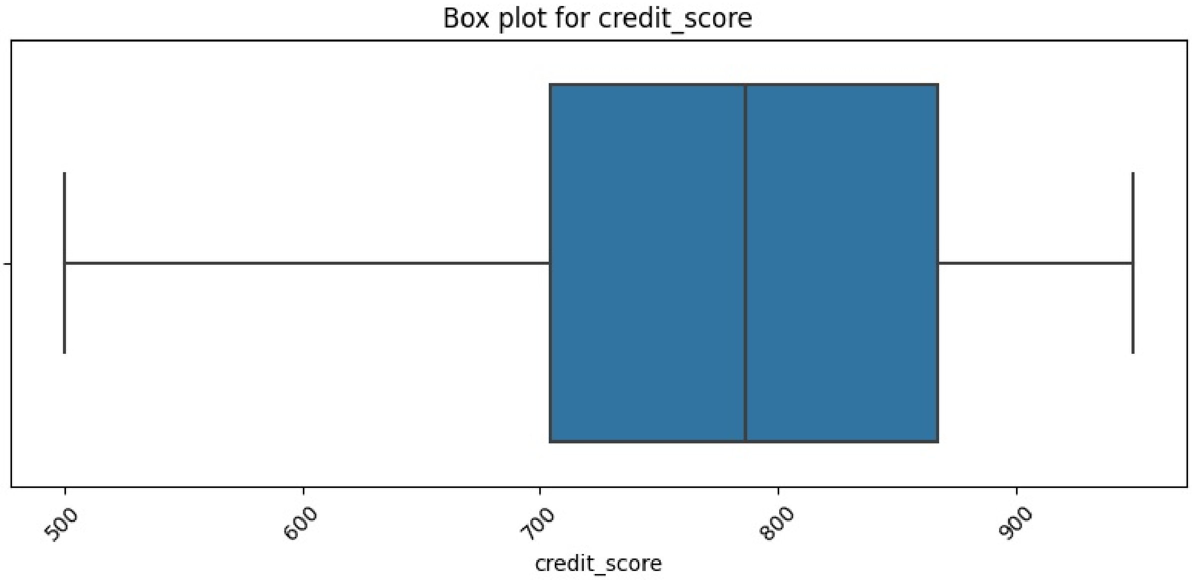

Box plot for net_yearly_income
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
net_yearly_income
1e8


Box plot for no_of_days_employed
0
50000
100000
150000
200000
250000
300000
350000
no_of_days_employed


Box plot for yearly_debt_payments
0
50000
100000
150000
200000
250000
300000
yearly_debt_payments







In [32]:
```
#owns car-randomly impute
# no of children- randomly impute
# total family members-randomly impute
# migrant worker-randomly impute
```

In [33]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [34]:
```
x=data.drop(columns=['credit_card_default'])
```

```
x
```

Out[34]:

	age	gender	owns_car	owns_house	no_of_children	net_yearly_income	no_of_days_employed	occupation_type	total_fam
0	39	F	N	Y	0.0	160503.35	1154.0	Cooking staff	
1	32	F	N	Y	1.0	310268.73	239.0	Core staff	
2	27	M	Y	N	0.0	264593.49	4014.0	Laborers	
3	24	F	N	Y	0.0	170396.97	4189.0	Medicine staff	
4	44	F	N	Y	0.0	222185.59	8438.0	Core staff	
...	...	...	...	...	...	...	...	...	
56905	29	M	Y	N	0.0	174277.82	815.0	Managers	
56906	38	F	Y	Y	1.0	78132.67	3479.0	Unknown	
56907	52	F	N	Y	0.0	220697.50	365249.0	Unknown	
56909	35	F	N	Y	1.0	160382.15	9322.0	Unknown	
56910	48	F	N	Y	0.0	210916.56	365245.0	Unknown	

45527 rows × 16 columns

In [35]:

```
y=data['credit_card_default']
y
```

Out[35]:

```
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
        ...
56905    0.0
56906    0.0
56907    0.0
56909    0.0
56910    0.0
Name: credit_card_default, Length: 45527, dtype: float64
```

In [ ]:

```
# Split training and test data
```

In [36]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
x_train
```

Out[36]:

	age	gender	owns_car	owns_house	no_of_children	net_yearly_income	no_of_days_employed	occupation_type	total_fam
29561	42	M	N	Y	0.0	138896.56	377.0	High skill tech staff	
9590	52	F	N	Y	0.0	68291.13	365251.0	Unknown	
14554	29	M	Y	Y	3.0	198097.86	438.0	Cooking staff	
17970	48	F	N	Y	0.0	118635.11	875.0	Private service staff	
34335	51	F	N	N	0.0	99588.95	NaN	Sales staff	
...	...	...	...	...	...	...	...	...	
14143	41	F	Y	N	1.0	257661.00	3569.0	Core staff	
55899	50	M	N	N	0.0	252562.84	999.0	Sales staff	
47671	51	F	Y	Y	2.0	186719.73	1173.0	Core staff	
1107	46	F	N	Y	2.0	134011.93	1978.0	Laborers	
19825	27	F	Y	N	0.0	167869.35	7160.0	High skill tech staff	

36421 rows × 16 columns

In [37]:

```
x_train['owns_car_imputed']=x_train['owns_car']
```

In [38]:

```
x_train['no_of_children_imputed']=x_train['no_of_children']
```

In [39]:

```
x_train['total_family_members_imputed']=x_train['total_family_members']
```

In [40]:

```
x_train['migrant_worker_imputed']=x_train['migrant_worker']
```

```
In [ ]: # Randomly filling values in categorical columns
```

```
In [41]: x_train['owns_car_imputed'][x_train['owns_car_imputed'].isnull()]=x_train['owns_car'].dropna().sample(x_train['
```

```
In [42]: x_train['no_of_children_imputed'][x_train['no_of_children_imputed'].isnull()]=x_train['no_of_children'].dropna(
```

```
In [43]: x_train['total_family_members_imputed'][x_train['total_family_members_imputed'].isnull()]=x_train['total_family_
```

```
In [44]: x_train['migrant_worker_imputed'][x_train['migrant_worker_imputed'].isnull()]=x_train['migrant_worker'].dropna(
```

```
In [ ]: # calculatioe median of columns where there was missing values
```

```
In [45]: median_no_of_days_employed=x_train['no_of_days_employed'].median()
         median_yearly_debt_payments=x_train['yearly_debt_payments'].median()
         median_credit_score=x_train['credit_score'].median()
```

```
In [46]: print(median_no_of_days_employed)
         print(median_yearly_debt_payments)
         print(median_credit_score)
```

```
2228.0
29096.015
786.0
```

Impute Missing Values

```
In [47]: x_train['median_no_of_days_employed']=x_train['no_of_days_employed'].fillna(median_no_of_days_employed)
```

```
In [48]: x_train['median_yearly_debt_payments']=x_train['yearly_debt_payments'].fillna(median_yearly_debt_payments)
```

```
In [49]: x_train['median_credit_score']=x_train['credit_score'].fillna(median_credit_score)
```

```
In [50]: x_train
```

Out[50]:

	age	gender	owns_car	owns_house	no_of_children	net_yearly_income	no_of_days_employed	occupation_type	total_fam
29561	42	M	N	Y	0.0	138896.56	377.0	High skill tech staff	
9590	52	F	N	Y	0.0	68291.13	365251.0	Unknown	
14554	29	M	Y	Y	3.0	198097.86	438.0	Cooking staff	
17970	48	F	N	Y	0.0	118635.11	875.0	Private service staff	
34335	51	F	N	N	0.0	99588.95	NaN	Sales staff	
...	...	...	...	...	...	...	...	...	
14143	41	F	Y	N	1.0	257661.00	3569.0	Core staff	
55899	50	M	N	N	0.0	252562.84	999.0	Sales staff	
47671	51	F	Y	Y	2.0	186719.73	1173.0	Core staff	
1107	46	F	N	Y	2.0	134011.93	1978.0	Laborers	
19825	27	F	Y	N	0.0	167869.35	7160.0	High skill tech staff	

36421 rows × 23 columns

```
In [52]: x_train.isnull().sum()
```

Out[52]:

```
age                              0
gender                           0
owns_car                       457
owns_house                       0
no_of_children                 639
net_yearly_income                0
no_of_days_employed            358
occupation_type                  0
total_family_members            66
migrant_worker                  69
yearly_debt_payments            69
credit_limit                     0
credit_limit_used(%)             0
credit_score                     7
prev_defaults                    0
default_in_last_6months          0
owns_car_imputed                 0
no_of_children_imputed           0
total_family_members_imputed     0
migrant_worker_imputed           0
median_no_of_days_employed       0
median_yearly_debt_payments      0
median_credit_score              0
dtype: int64
```

In [54]:

```
x_test['owns_car_imputed']=x_test['owns_car']
```

In [55]:

```
x_test['no_of_children_imputed']=x_test['no_of_children']
```

In [56]:

```
x_test['total_family_members_imputed']=x_test['total_family_members']
```

In [57]:

```
x_test['migrant_worker_imputed']=x_test['migrant_worker']
```

In [58]:

```
x_test['owns_car_imputed'][x_test['owns_car_imputed'].isnull()]=x_test['owns_car'].dropna().sample(x_test['owns_
```

In [59]:

```
x_test['no_of_children_imputed'][x_test['no_of_children_imputed'].isnull()]=x_test['no_of_children'].dropna().sa
```

In [60]:

```
x_test['total_family_members_imputed'][x_test['total_family_members_imputed'].isnull()]=x_test['total_family_mem
```

In [61]:

```
x_test['migrant_worker_imputed'][x_test['migrant_worker_imputed'].isnull()]=x_test['migrant_worker'].dropna().sa
```

In [62]:

```
x_test['median_no_of_days_employed']=x_test['no_of_days_employed'].fillna(median_no_of_days_employed)
```

In [63]:

```
x_test['median_yearly_debt_payments']=x_test['yearly_debt_payments'].fillna(median_yearly_debt_payments)
```

In [64]:

```
x_test['median_credit_score']=x_test['credit_score'].fillna(median_credit_score)
```

In [65]:

```
x_test
```

Out[65]:

	age	gender	owns_car	owns_house	no_of_children	net_yearly_income	no_of_days_employed	occupation_type	total_fam
42606	29	M	N	Y	1.0	77844.14	226.0	Drivers	
33907	34	M	Y	N	0.0	270526.52	2482.0	Laborers	
46950	42	M	Y	Y	0.0	200185.84	755.0	Laborers	
41888	29	F	Y	Y	0.0	142799.44	887.0	Laborers	
35033	33	F	N	Y	0.0	171308.07	11496.0	Laborers	
...	...	...	...	...	...	...	...	...	
13813	38	F	N	N	0.0	224381.60	5472.0	Laborers	
9613	43	F	Y	Y	0.0	294502.61	2852.0	Accountants	
16022	54	M	Y	N	0.0	194556.05	365243.0	Unknown	
21126	51	F	Y	Y	0.0	84777.44	2244.0	Sales staff	
50427	55	M	Y	Y	0.0	172307.61	5282.0	Laborers	

9106 rows × 23 columns

In [66]:

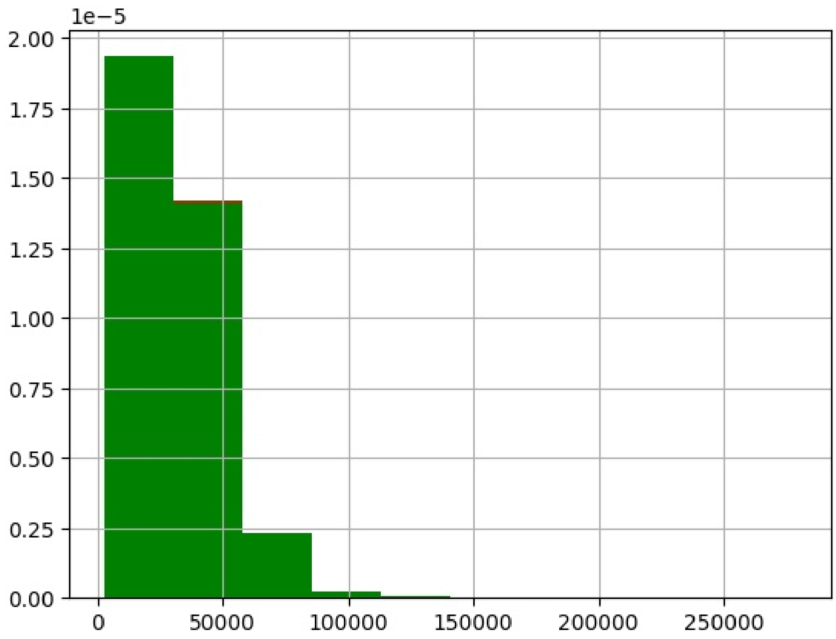
```
x_test.isnull().sum()
```

```
Out[66]: age                            0
         gender                         0
         owns_car                      90
         owns_house                     0
         no_of_children               135
         net_yearly_income              0
         no_of_days_employed          105
         occupation_type                0
         total_family_members          17
         migrant_worker                18
         yearly_debt_payments          26
         credit_limit                   0
         credit_limit_used(%)           0
         credit_score                   1
         prev_defaults                  0
         default_in_last_6months        0
         owns_car_imputed               0
         no_of_children_imputed         0
         total_family_members_imputed   0
         migrant_worker_imputed         0
         median_no_of_days_employed     0
         median_yearly_debt_payments    0
         median_credit_score            0
         dtype: int64
```

```
In [ ]: # Distribution of column yearly_debt_payments before and after the imputing the values
```

```
In [67]: fig=plt.figure()
         ax=fig.add_subplot(111)
         x_train['yearly_debt_payments'].hist(ax=ax,density=True,color='red')
         x_train['median_yearly_debt_payments'].hist(ax=ax,density=True,color='green')
```

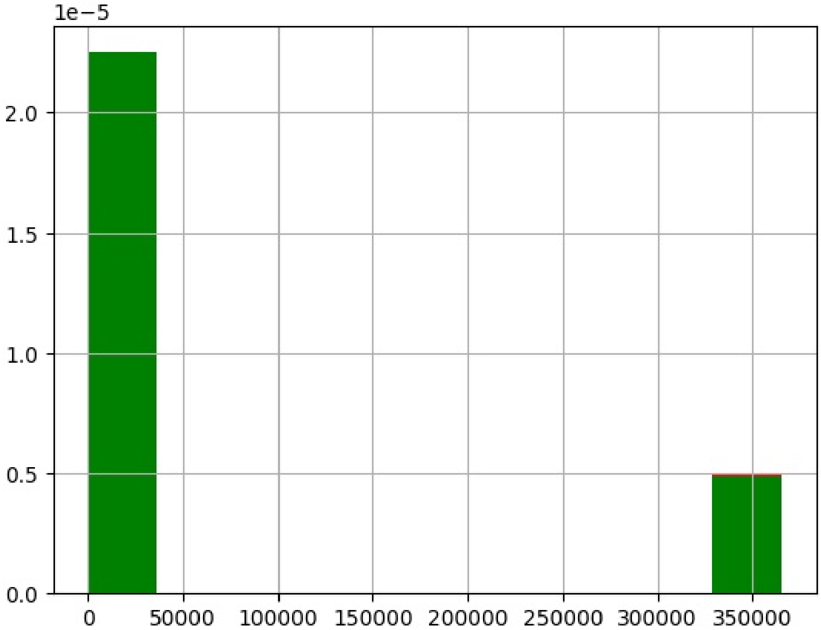
```
Out[67]: <Axes: >
```




```
In [68]: # Distribution of column no. of days_employed before and after the imputing the values
```

```
In [69]: fig=plt.figure()
         ax=fig.add_subplot(111)
         x_train['no_of_days_employed'].hist(ax=ax,density=True,color='red')
         x_train['median_no_of_days_employed'].hist(ax=ax,density=True,color='green')
```

```
Out[69]: <Axes: >
```



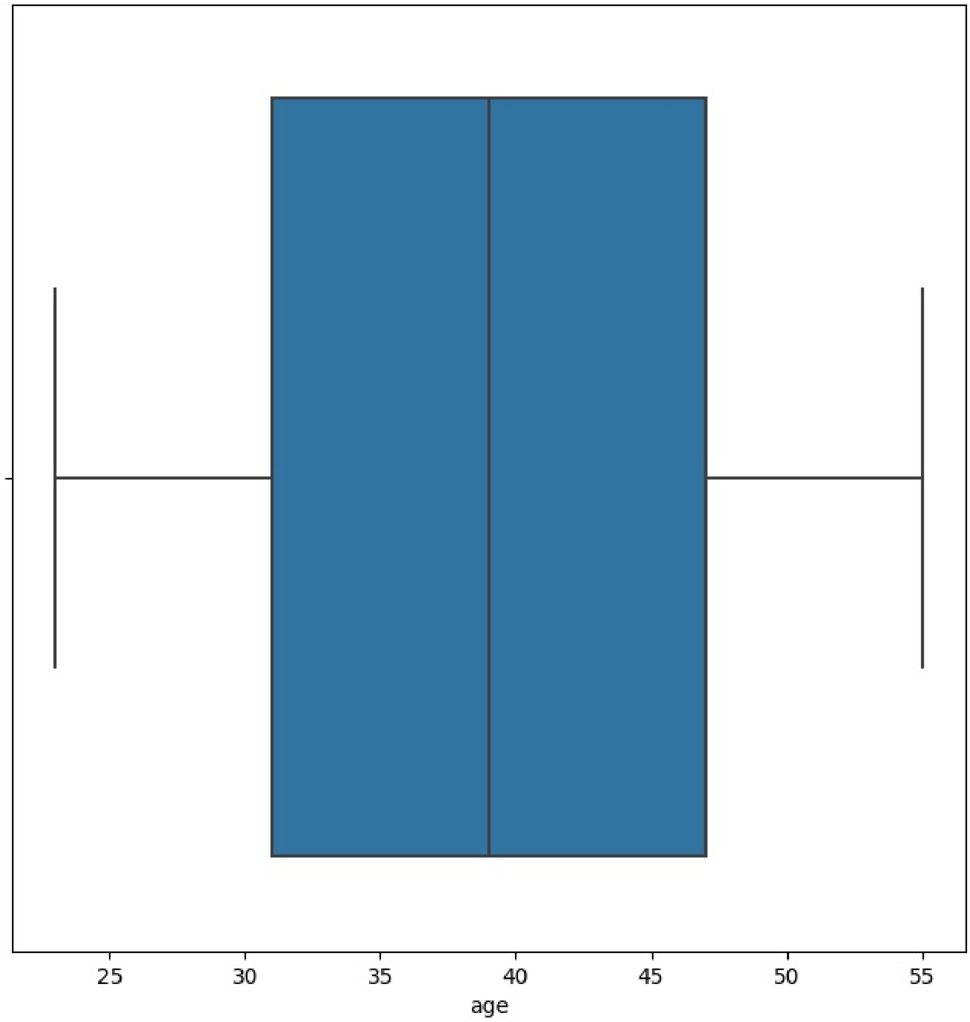
In [70]:
```
x_train=x_train.drop(columns=['owns_car','no_of_children','no_of_days_employed','total_family_members','migrant_
```

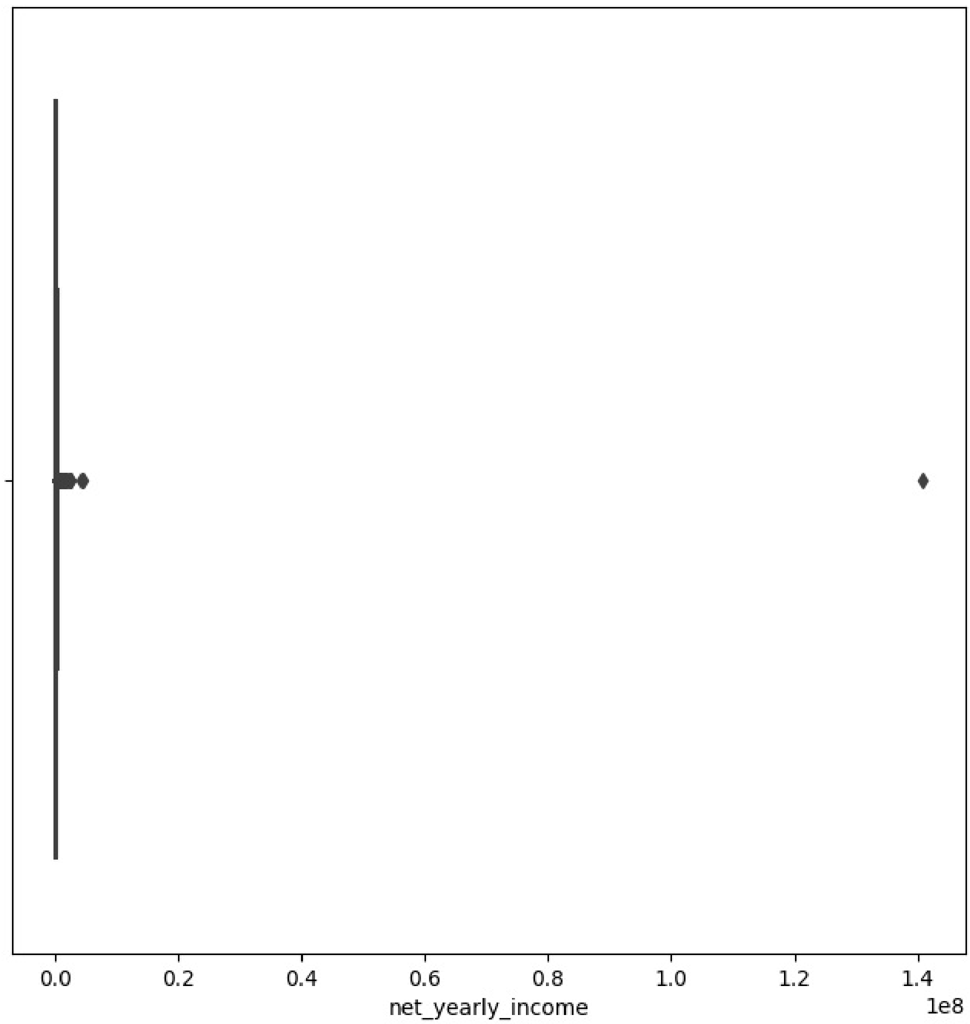
In [71]:
```
x_test=x_test.drop(columns=['owns_car','no_of_children','no_of_days_employed','total_family_members','migrant_wo
```

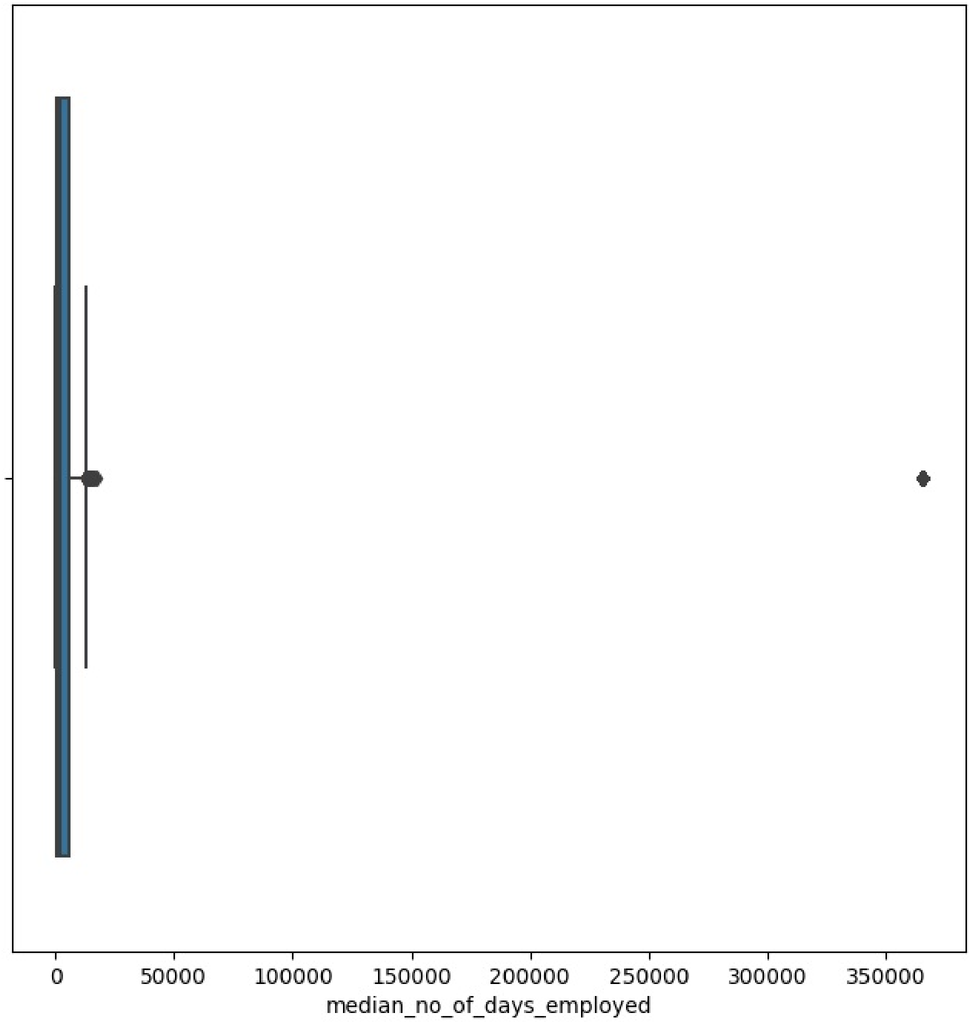
In [72]:
```
list2=['age','net_yearly_income','median_no_of_days_employed','median_yearly_debt_payments','credit_limit','cre
```

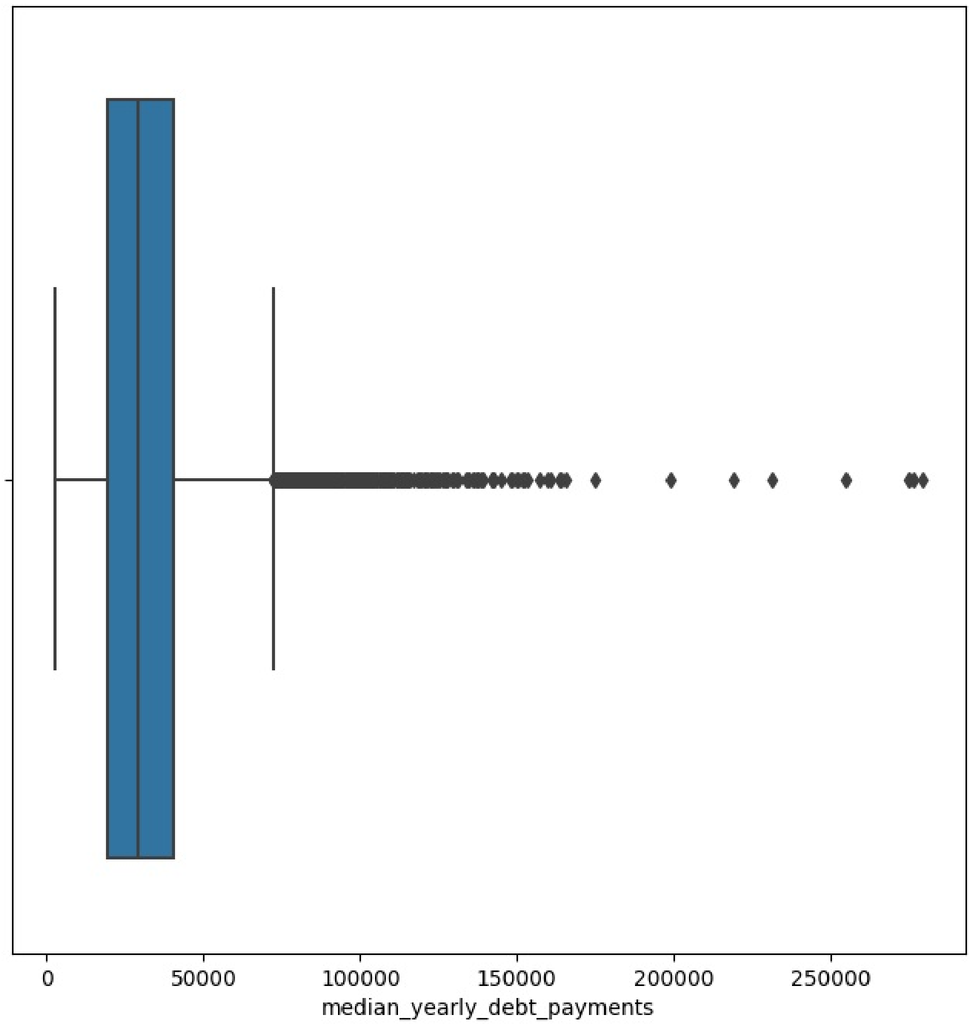
In [ ]:
```
# Number of outliers in training data
```

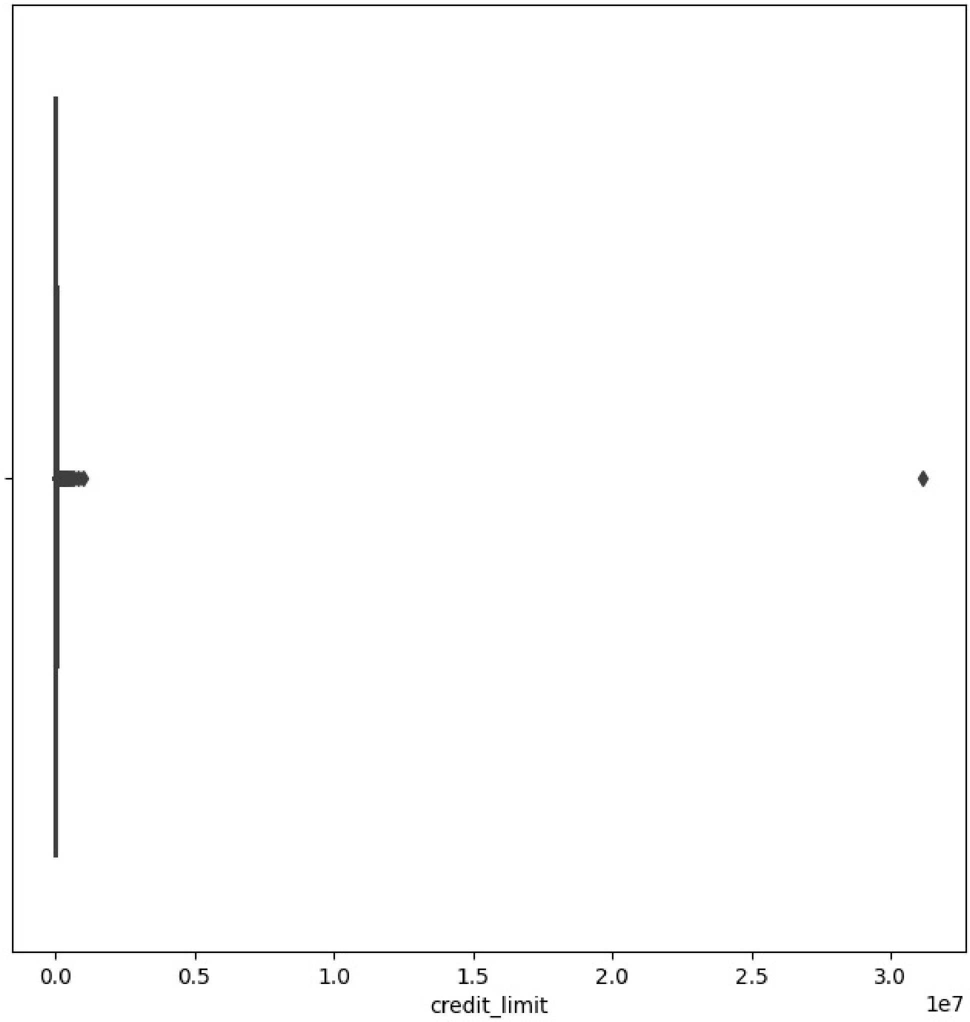
In [73]:
```
for col in list2:
    plt.figure(figsize=(8,8))
    sns.boxplot(data=x_train,x=col)
```

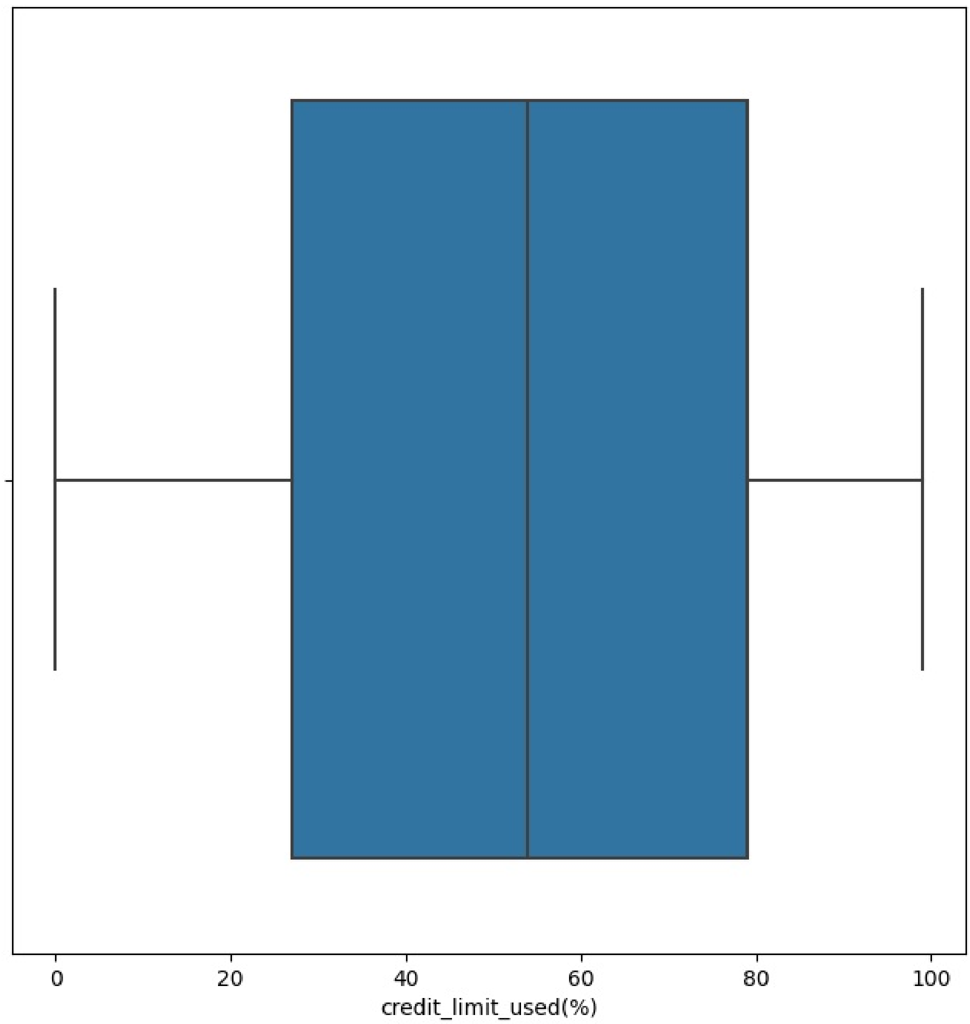

25
30
35
40
45
50
55
age

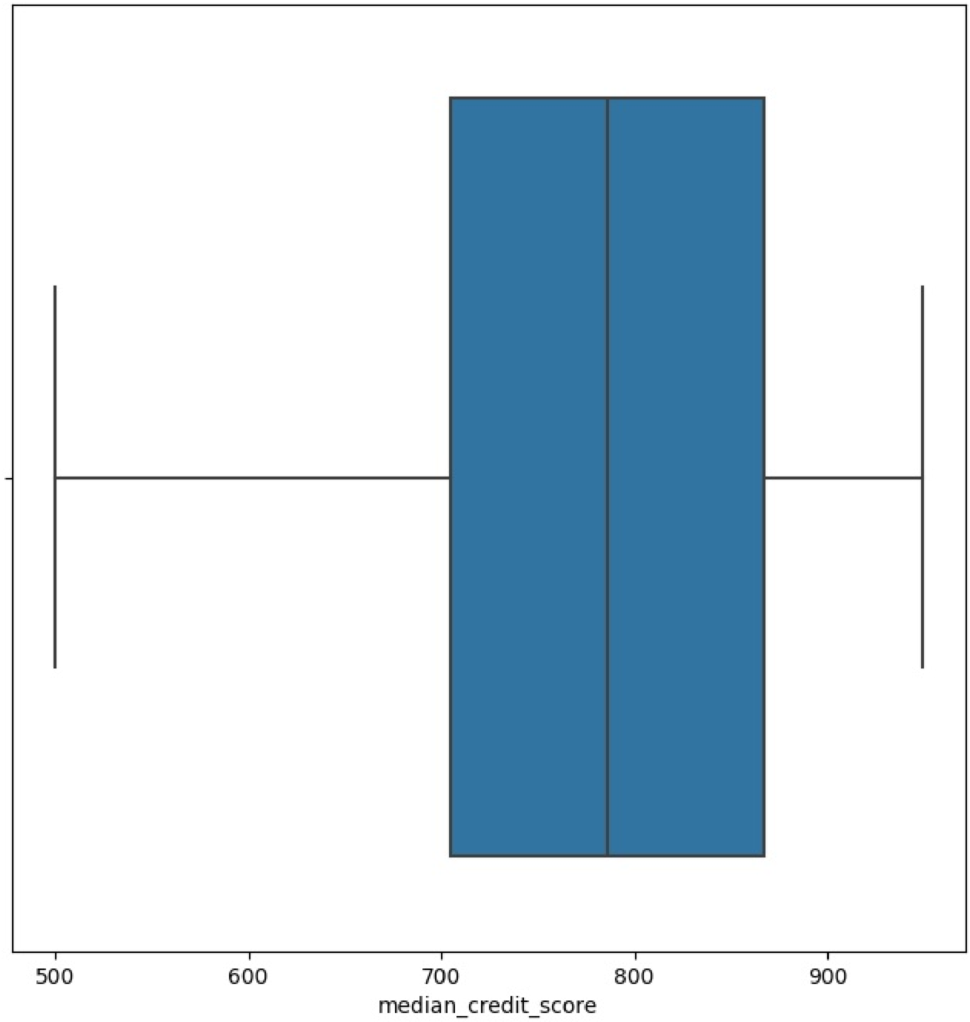

0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e8
net_yearly_income


0
50000
100000
150000
200000
250000
300000
350000
median_no_of_days_employed


0
50000
100000
150000
200000
250000
median_yearly_debt_payments


0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e7
credit_limit


0
20
40
60
80
100
credit_limit_used(%)



In [ ]:
```
# Finding total number of outliers in numerical columns
```

In [74]:
```
list3=['median_no_of_days_employed','median_yearly_debt_payments','credit_limit','net_yearly_income']
```

In [75]:
```
q1=x_train['median_no_of_days_employed'].quantile(0.25)
q3=x_train['median_no_of_days_employed'].quantile(0.75)
l1=q1-1.5*(q3-q1)
l2=q3+1.5*(q3-q1)
```

In [76]:
```
outliers=x_train[(x_train['median_no_of_days_employed']<l1)|(x_train['median_no_of_days_employed']>l2)]
num_outliers=len(outliers)
num_outliers
```

Out[76]: 6627

In [77]:
```
p1=x_train['median_yearly_debt_payments'].quantile(0.25)
p3=x_train['median_yearly_debt_payments'].quantile(0.75)
l11=p1-1.5*(p3-p1)
l12=p3+1.5*(p3-p1)
```

In [78]:
```
outliers1=x_train[(x_train['median_yearly_debt_payments']<l11)|(x_train['median_yearly_debt_payments']>l12)]
num_outliers1=len(outliers1)
num_outliers1
```

Out[78]: 853

In [79]:
```
r1=x_train['credit_limit'].quantile(0.25)
r3=x_train['credit_limit'].quantile(0.75)
l111=r1-1.5*(r3-r1)
l122=r3+1.5*(r3-r1)
```

In [80]:
```
outliers2=x_train[(x_train['credit_limit']<l111)|(x_train['credit_limit']>l122)]
num_outliers2=len(outliers2)
num_outliers2
```

Out[80]: 1565

In [81]:
```
s1=x_train['net_yearly_income'].quantile(0.25)
s3=x_train['net_yearly_income'].quantile(0.75)
l1111=s1-1.5*(s3-s1)
```

```
l1222=s3+1.5*(s3-s1)
```

In [82]:
```
outliers3=x_train[(x_train['net_yearly_income']<l1111)|(x_train['net_yearly_income']>l1222)]
num_outliers3=len(outliers3)
num_outliers3
```

Out[82]: 1438

In [ ]:
```
# Capping the outliers
```

In [83]:
```
x_train['median_no_of_days_employed']=np.where(x_train['median_no_of_days_employed']>l2,l2,
                                              np.where(x_train['median_no_of_days_employed']<l1,l1,
                                              x_train['median_no_of_days_employed'] ))
```

In [84]:
```
x_train['median_yearly_debt_payments']=np.where(x_train['median_yearly_debt_payments']>l12,l12,
                                              np.where(x_train['median_yearly_debt_payments']<l11,l11,
                                              x_train['median_yearly_debt_payments'] ))
```

In [85]:
```
x_train['credit_limit']=np.where(x_train['credit_limit']>l122,l122,
                                              np.where(x_train['credit_limit']<l111,l111,
                                              x_train['credit_limit'] ))
```

In [86]:
```
x_train['net_yearly_income']=np.where(x_train['net_yearly_income']>l1222,l1222,
                                              np.where(x_train['net_yearly_income']<l1111,l1111,
                                              x_train['net_yearly_income'] ))
```

In [87]:
```
x_train
```

Out[87]:

	age	gender	owns_house	net_yearly_income	occupation_type	credit_limit	credit_limit_used(%)	prev_defaults	default_in_l
29561	42	M	Y	138896.56	High skill tech staff	33257.60	58	0	
9590	52	F	Y	68291.13	Unknown	10666.87	87	1	
14554	29	M	Y	198097.86	Cooking staff	77343.08	35	0	
17970	48	F	Y	118635.11	Private service staff	15570.15	77	0	
34335	51	F	N	99588.95	Sales staff	30094.61	59	0	
...	...	...	...	...	...	...	...	...	
14143	41	F	N	257661.00	Core staff	40832.94	47	0	
55899	50	M	N	252562.84	Sales staff	43322.92	21	0	
47671	51	F	Y	186719.73	Core staff	48751.57	55	0	
1107	46	F	Y	134011.93	Laborers	42045.06	56	0	
19825	27	F	N	167869.35	High skill tech staff	30636.27	74	0	

36421 rows × 16 columns

In [88]:
```
mean=x_train['credit_limit'].mean()
std=x_train['credit_limit'].std()
(33257.60-mean)/std
```

Out[88]: -0.34757573501205324

In [89]:
```
x_test['median_no_of_days_employed']=np.where(x_test['median_no_of_days_employed']>l2,l2,
                                              np.where(x_test['median_no_of_days_employed']<l1,l1,
                                              x_test['median_no_of_days_employed'] ))
```

In [90]:
```
x_test['median_yearly_debt_payments']=np.where(x_test['median_yearly_debt_payments']>l12,l12,
                                              np.where(x_test['median_yearly_debt_payments']<l11,l11,
                                              x_test['median_yearly_debt_payments'] ))
```

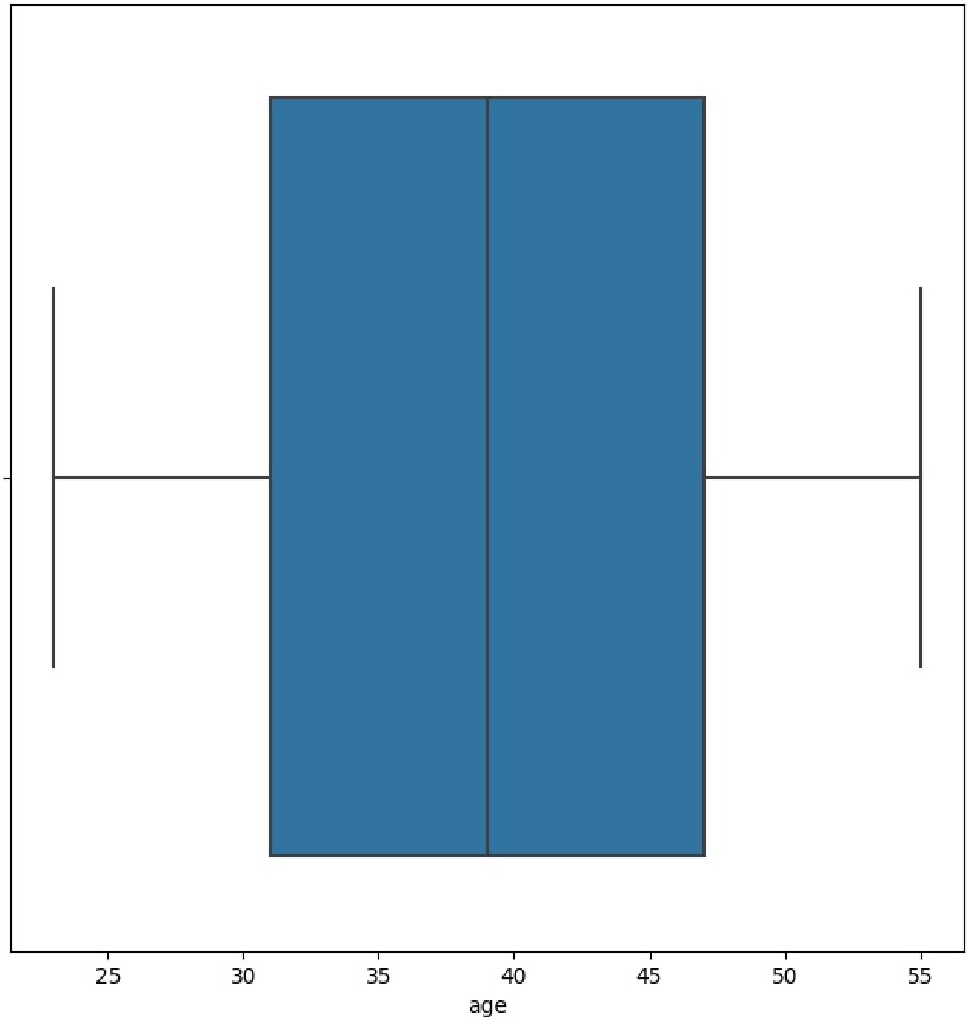
In [91]:
```
x_test['credit_limit']=np.where(x_test['credit_limit']>l122,l122,
                                              np.where(x_test['credit_limit']<l111,l111,
                                              x_test['credit_limit'] ))
```

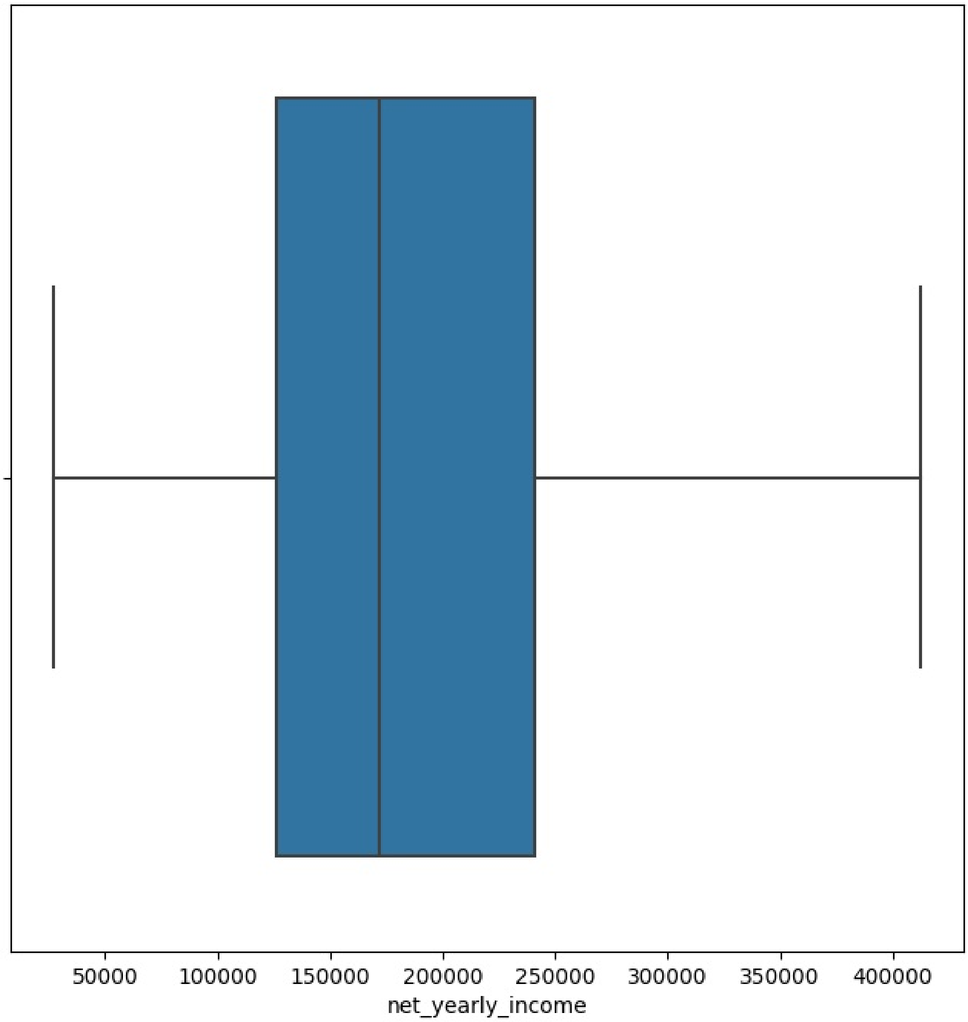
In [92]:
```
x_test['net_yearly_income']=np.where(x_test['net_yearly_income']>l1222,l1222,
                                              np.where(x_test['net_yearly_income']<l1111,l1111,
                                              x_test['net_yearly_income'] ))
```

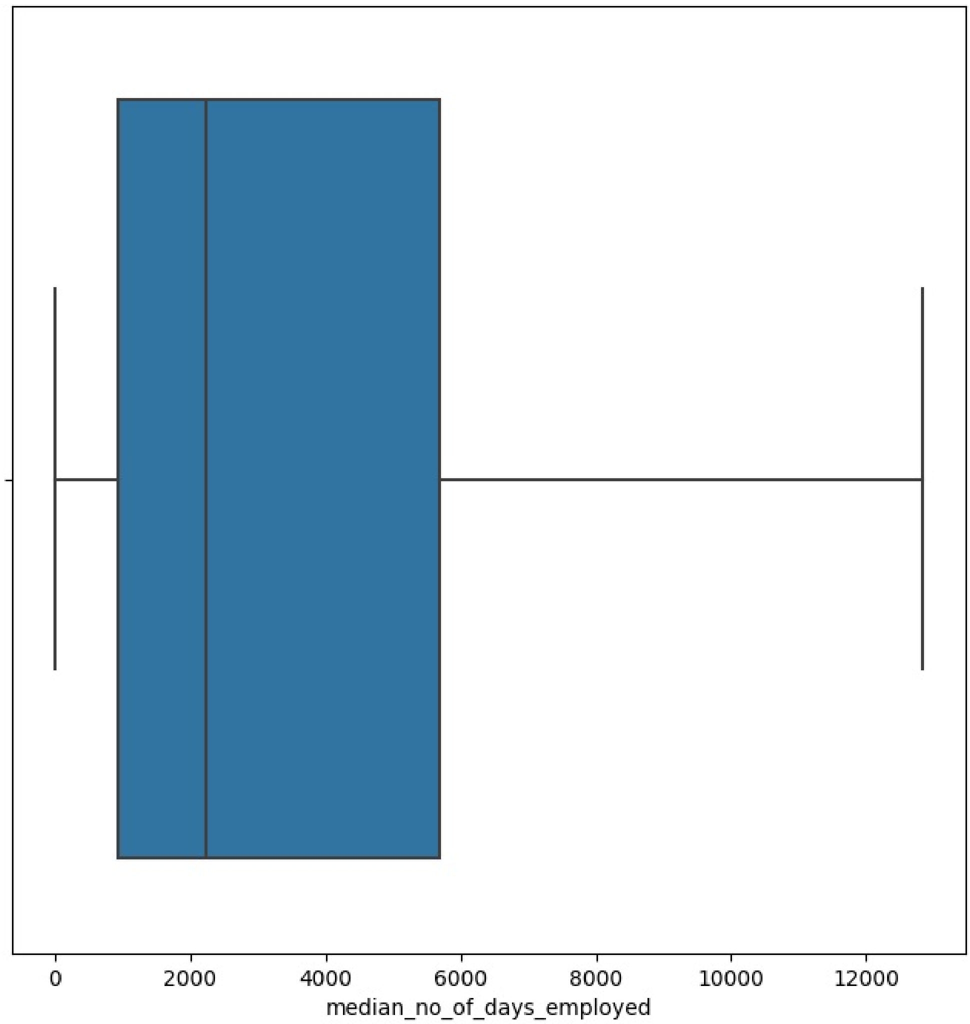
In [ ]:
```
# Now,training data is free from outliers
```

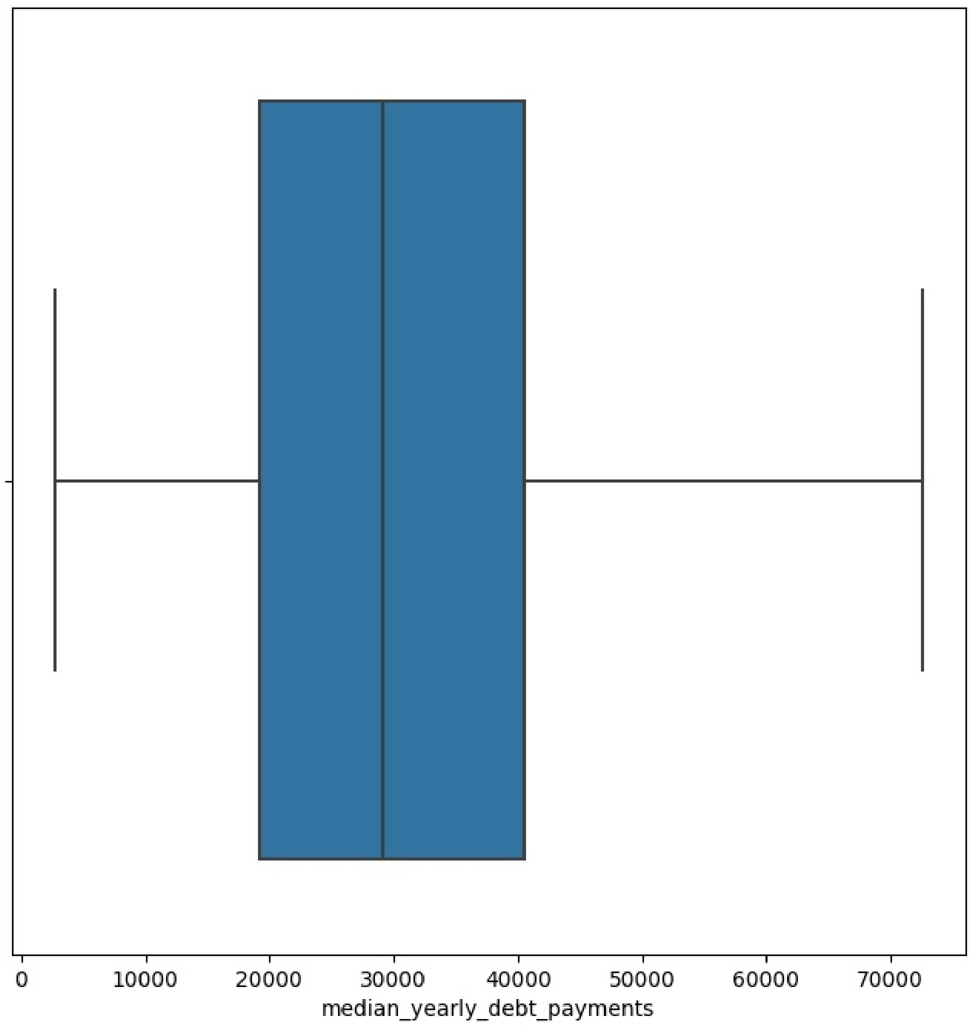
In [93]:
```
for col in list2:
```

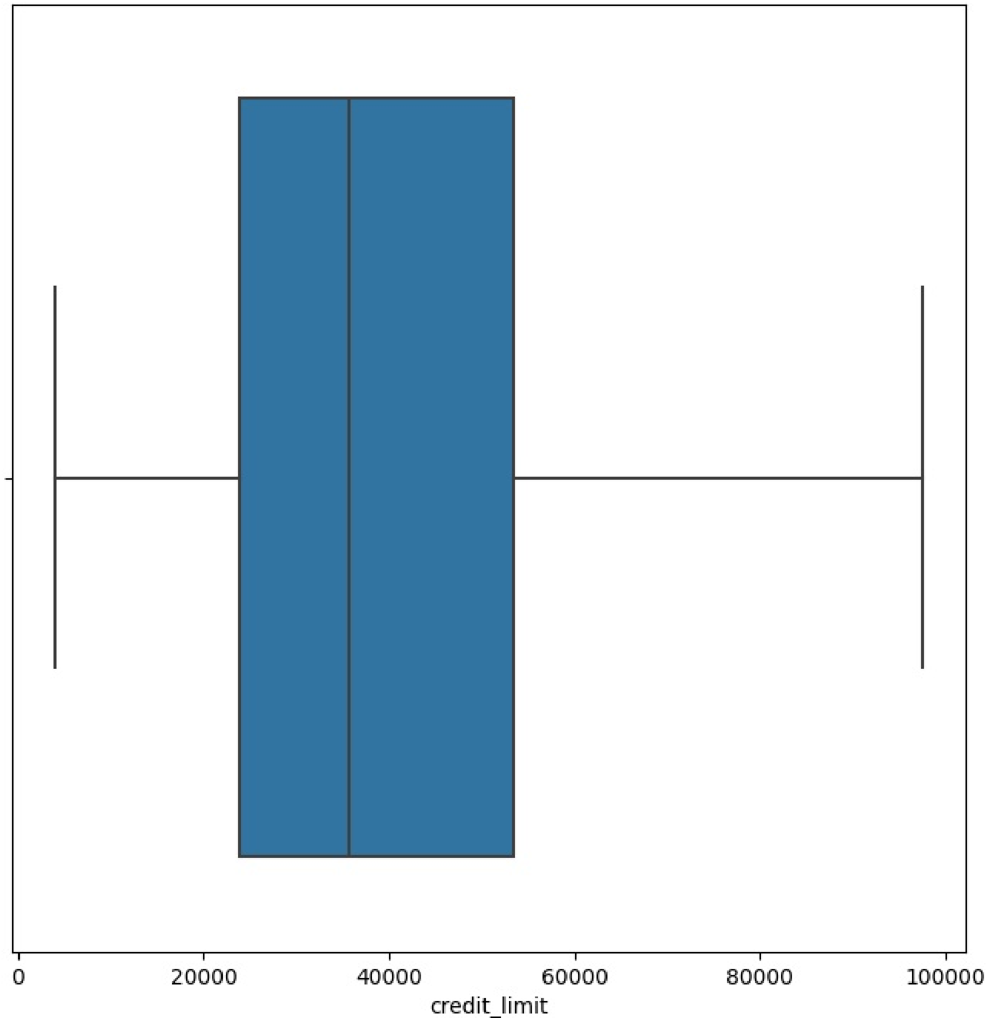
```
    plt.figure(figsize=(8,8))
    sns.boxplot(data=x_train,x=col)
```

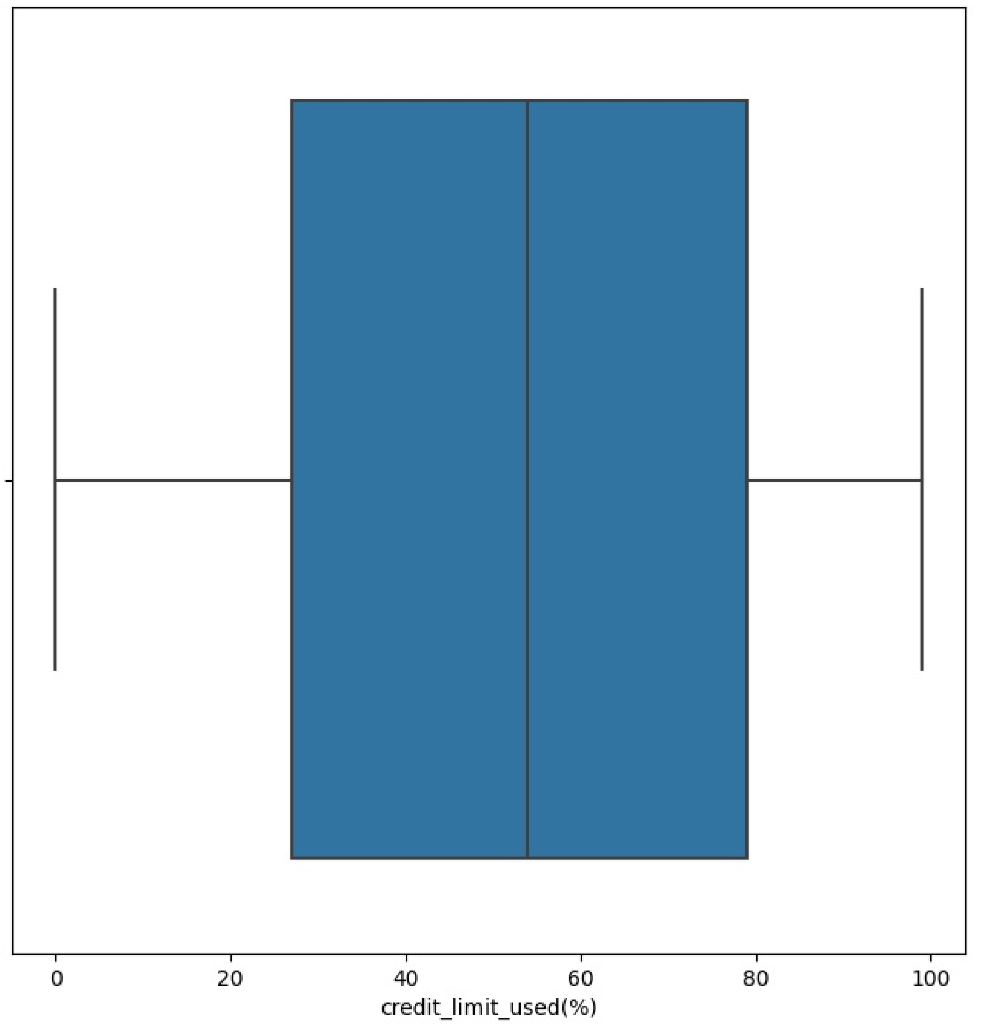


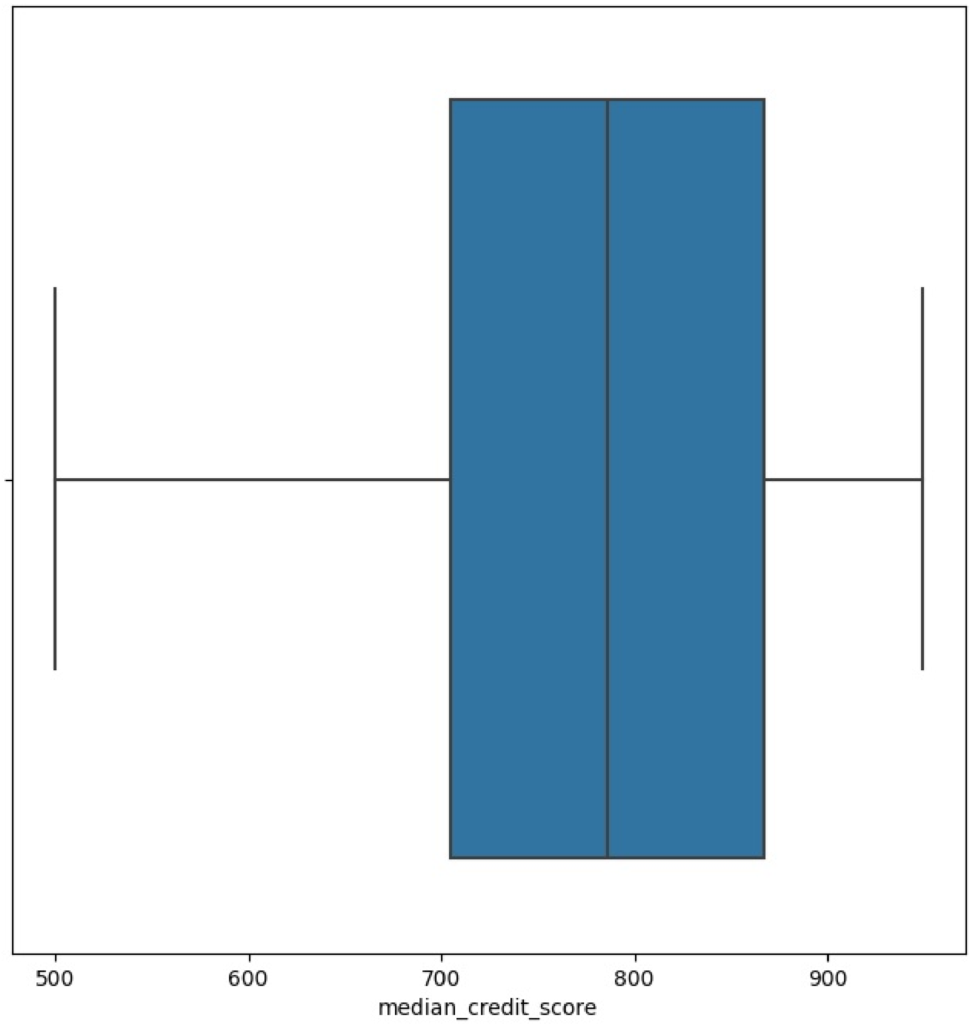



0
2000
4000
6000
8000
10000
12000
median_no_of_days_employed


0
10000
20000
30000
40000
50000
60000
70000
median_yearly_debt_payments


0
20000
40000
60000
80000
100000
credit_limit


0
20
40
60
80
100
credit_limit_used(%)



In [94]:
```
# We use column transformer to use one hot encoding ,standard scaling on the columns
```

In [95]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
step1=ColumnTransformer(transformers=[
    ('trf1',OneHotEncoder(sparse=False,drop='first'),['gender','owns_house','occupation_type','owns_car_imputed
    ('trf2',StandardScaler(),['age','net_yearly_income','credit_limit','credit_limit_used(%)','median_no_of_days
                              'median_yearly_debt_payments','median_credit_score'])],remainder='passthrough')
```

In [96]:
```
x_train1=step1.fit_transform(x_train)
x_test1=step1.fit_transform(x_test)
```

In [97]:
```
x_train1.shape
```

Out[97]:
```
(36421, 33)
```

In [98]:
```
x_test1.shape
```

Out[98]:
```
(9106, 33)
```

In [99]:
```
x_train1
```

Out[99]:
```
array([[1., 1., 0., ..., 0., 2., 1.],
       [0., 1., 0., ..., 0., 2., 0.],
       [1., 1., 0., ..., 3., 5., 0.],
       ...,
       [0., 1., 0., ..., 2., 4., 0.],
       [0., 1., 0., ..., 2., 4., 1.],
       [0., 0., 0., ..., 0., 2., 0.]])
```

In [ ]:
```
# Creating numpy array in Dataframe
```

In [100…
```
x_train1 = pd.DataFrame(x_train1, columns=[
    'gender', 'owns_house', 'Cleaning staff_ohe', 'Cooking staff_ohe',
    'Core staff_ohe', 'Drivers_ohe', 'HR staff_ohe', 'High skill tech staff_ohe',
    'IT staff_ohe', 'Laborers_ohe', 'Low-skill Laborers_ohe', 'Managers_ohe',
    'Medicine staff _ohe', 'Private service staff_ohe', 'Realty agents_ohe',
    'Sales staff_ohe', 'Secretaries_ohe', 'Security staff_ohe', 'Unknown_ohe',
```

```
    'Waiters/barmen staff_ohe', 'owns_car_imputed', 'Age', 'net_yearly_income',
    'credit_limit', 'credit_limit_used(%)', 'median_no_of_days_employed',
    'median_yearly_debt_payments', 'median_credit_score', 'prev_defaults',
    'default_in_last_6months', 'no_of_children_imputed', 'total_family_members_imputed',
    'migrant_worker_imputed'
])
x_test1= pd.DataFrame(x_test1, columns=[
    'gender', 'owns_house', 'Cleaning staff_ohe', 'Cooking staff_ohe',
    'Core staff_ohe', 'Drivers_ohe', 'HR staff_ohe', 'High skill tech staff_ohe',
    'IT staff_ohe', 'Laborers_ohe', 'Low-skill Laborers_ohe', 'Managers_ohe',
    'Medicine staff _ohe', 'Private service staff_ohe', 'Realty agents_ohe',
    'Sales staff_ohe', 'Secretaries_ohe', 'Security staff_ohe', 'Unknown_ohe',
    'Waiters/barmen staff_ohe', 'owns_car_imputed', 'Age', 'net_yearly_income',
    'credit_limit', 'credit_limit_used(%)', 'median_no_of_days_employed',
    'median_yearly_debt_payments', 'median_credit_score', 'prev_defaults',
    'default_in_last_6months', 'no_of_children_imputed', 'total_family_members_imputed',
    'migrant_worker_imputed'
])
```

In [101…

```
x_train1
```

Out[101…

	gender	owns_house	Cleaning staff_ohe	Cooking staff_ohe	Core staff_ohe	Drivers_ohe	HR staff_ohe	High skill tech staff_ohe	IT staff_ohe	Laborers_ohe	...	credit_
0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	-0.34
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.33
2	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.58
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.12
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.48
...	...	...	...	...	...	...	...	...	...	...	...	
36416	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	-0.01
36417	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.09
36418	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.33
36419	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.03
36420	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	-0.46

36421 rows × 33 columns

In [102…

```
x_test1
```

Out[102…

	gender	owns_house	Cleaning staff_ohe	Cooking staff_ohe	Core staff_ohe	Drivers_ohe	HR staff_ohe	High skill tech staff_ohe	IT staff_ohe	Laborers_ohe	...	credit_l
0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	-0.855
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.546
2	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	1.031
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.304
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	-0.276
...	...	...	...	...	...	...	...	...	...	...	...	
9101	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	-0.475
9102	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.101
9103	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.546
9104	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.982
9105	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	-0.157

9106 rows × 33 columns

In [ ]:

```
# Correlation corresponding to each column
```

In [157…

```
x_train1.corr()
```

Out[157…

	gender	owns_house	Cleaning staff_ohe	Cooking staff_ohe	Core staff_ohe	Drivers_ohe	HR staff_ohe	High skill tech staff_ohe	IT staff_oh
gender	1.000000	-0.038969	-0.071457	-0.076113	-0.090676	0.323541	-0.023323	0.013460	0.02562
owns_house	-0.038969	1.000000	0.011700	0.000568	-0.018601	-0.007725	-0.008924	-0.011933	-0.01441
Cleaning staff_ohe	-0.071457	0.011700	1.000000	-0.017513	-0.038453	-0.031219	-0.004992	-0.024279	-0.00445
Cooking staff_ohe	-0.076113	0.000568	-0.017513	1.000000	-0.043841	-0.035594	-0.005692	-0.027681	-0.00507
Core staff_ohe	-0.090676	-0.018601	-0.038453	-0.043841	1.000000	-0.078151	-0.012497	-0.060778	-0.01115
Drivers_ohe	0.323541	-0.007725	-0.031219	-0.035594	-0.078151	1.000000	-0.010146	-0.049344	-0.00905
HR staff_ohe	-0.023323	-0.008924	-0.004992	-0.005692	-0.012497	-0.010146	1.000000	-0.007891	-0.00144
High skill tech staff_ohe	0.013460	-0.011933	-0.024279	-0.027681	-0.060778	-0.049344	-0.007891	1.000000	-0.00704
IT staff_ohe	0.025622	-0.014413	-0.004455	-0.005079	-0.011153	-0.009055	-0.001448	-0.007042	1.00000
Laborers_ohe	0.233798	-0.016315	-0.058151	-0.066299	-0.145567	-0.118183	-0.018899	-0.091911	-0.01686
Low-skill Laborers_ohe	0.085059	0.002016	-0.010550	-0.012028	-0.026410	-0.021441	-0.003429	-0.016675	-0.00306
Managers_ohe	0.066525	-0.008611	-0.033589	-0.038296	-0.084083	-0.068265	-0.010917	-0.053090	-0.00974
Medicine staff _ohe	-0.113130	0.006214	-0.021049	-0.023998	-0.052690	-0.042778	-0.006841	-0.033269	-0.00610
Private service staff_ohe	-0.055941	0.000872	-0.011161	-0.012725	-0.027940	-0.022684	-0.003628	-0.017642	-0.00323
Realty agents_ohe	-0.030707	-0.007182	-0.005705	-0.006504	-0.014281	-0.011594	-0.001854	-0.009017	-0.00165
Sales staff_ohe	-0.167191	0.002493	-0.042394	-0.048335	-0.106125	-0.086161	-0.013778	-0.067007	-0.01229
Secretaries_ohe	-0.038655	-0.011723	-0.008024	-0.009148	-0.020085	-0.016307	-0.002608	-0.012682	-0.00232
Security staff_ohe	0.126231	0.003430	-0.018950	-0.021606	-0.047438	-0.038514	-0.006159	-0.029952	-0.00549
Unknown_ohe	-0.139163	0.037002	-0.083963	-0.095728	-0.210183	-0.170644	-0.027289	-0.132710	-0.02435
Waiters/barmen staff_ohe	-0.028440	-0.005385	-0.008412	-0.009591	-0.021057	-0.017096	-0.002734	-0.013295	-0.00244
owns_car_imputed	0.339876	0.006988	-0.055534	-0.040796	-0.013305	0.188736	-0.007055	0.025049	0.00511
Age	-0.006384	0.004196	-0.000682	0.000549	0.002478	0.006773	0.004424	0.003471	0.00502
net_yearly_income	0.205606	0.008054	-0.057451	-0.046924	0.023082	0.069879	0.005898	0.039645	0.02166
credit_limit	0.164991	0.006594	-0.041530	-0.038972	0.017261	0.059509	0.011005	0.028962	0.02710
credit_limit_used(%)	0.020202	-0.000174	0.000609	0.002277	-0.008966	0.019797	-0.005652	-0.008768	-0.00090
median_no_of_days_employed	-0.172859	0.065414	-0.057692	-0.064057	-0.098522	-0.130300	-0.015183	-0.066566	-0.01379
median_yearly_debt_payments	0.087758	-0.001952	-0.038478	-0.024048	0.020081	0.039738	-0.000285	0.035769	0.00860
median_credit_score	-0.031618	0.007007	-0.009341	-0.015557	0.010337	-0.023933	-0.000043	0.013281	0.00926
prev_defaults	0.048358	0.003110	0.007441	0.013772	-0.020126	0.030402	-0.003852	-0.016286	-0.00224
default_in_last_6months	0.042620	0.000222	0.007130	0.020038	-0.018616	0.021407	0.000296	-0.017015	-0.00464
no_of_children_imputed	0.054154	-0.002045	-0.007021	0.016825	0.063495	0.033808	-0.002640	0.018393	-0.00291
total_family_members_imputed	0.087168	0.008498	-0.021628	0.009973	0.057280	0.056306	0.001975	0.017959	-0.00795
migrant_worker_imputed	0.135540	-0.025150	-0.000767	0.012833	0.000870	0.070336	-0.006288	0.004127	0.00522

33 rows × 33 columns

In [105…

```
data['credit_card_default'].value_counts()
```

Out[105…

```
credit_card_default
0.0    41830
1.0     3697
Name: count, dtype: int64
```

In [106…

```
m=data['credit_score'].mean()
s=data['credit_score'].std()
print(m)
print(s)
print((941-m)/s)
```

```
782.7937564533491
100.61943741001976
1.5723228793455424
```

In [107…

```
from sklearn.metrics import classification_report,f1_score,mean_squared_error,roc_auc_score,confusion_matrix,pr
```

In [108…

```
from sklearn.model_selection import cross_val_score
```

In [109…
```
def evaluation(model,x1,y1,x2,y2):
    model.fit(x1,y1)
    y_train_pred=model.predict(x1)
    y_test_pred=model.predict(x2)
    print('-'*50)
    print(f"For training data confusion matrix can be given as")
    print(confusion_matrix(y1,y_train_pred))

    print(classification_report(y1,y_train_pred))
    print(f"accuracy score of training data is:{100*accuracy_score(y1,y_train_pred)}")

    print(f"F1 score of training data is {100*f1_score(y1,y_train_pred,average='macro')}")

    print(f"Precision score of training data is {100*precision_score(y1,y_train_pred)}")

    print(f"Recall score of training data is {100*recall_score(y1,y_train_pred)}")
    print('-'*50)

    print(f" for test data confusion matrix given as")
    print(confusion_matrix(y2,y_test_pred))
    print(classification_report(y2,y_test_pred))
    print(f"accuracy score of test data is:{100*accuracy_score(y2,y_test_pred)}")

    print(f"F1 score of test data is {100*f1_score(y2,y_test_pred,average='macro')}")

    print(f"Precision score of test data is {100*precision_score(y2,y_test_pred)}")

    print(f"Recall score of test data is {100*recall_score(y2,y_test_pred)}")
    print('-'*50)

    rmse=np.sqrt(mean_squared_error(y2,y_test_pred))
    print("RMSE is",rmse)

    roc_auc=roc_auc_score(y2,y_test_pred,average=None)
    print("ROC_AUC",roc_auc)

    print(f"Precision Cross Validation Score {cross_val_score(model,x1,y1,cv=5,scoring='precision').mean()}")
    print(f"Recall Cross Validation Score {cross_val_score(model,x1,y1,cv=5,scoring='recall').mean()}")
    print(f"Accuracy cross validation score {cross_val_score(model,x1,y1,cv=5,scoring='accuracy').mean()}")
```

Model Training

In [110…
```
from sklearn.linear_model import LogisticRegression
```

In [113…
```
from collections import Counter
from imblearn.ensemble import BalancedRandomForestClassifier
```

In [114…
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(random_state=42)
```

In [115…
```
from sklearn.model_selection import cross_val_score
```

In [116…
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
dt=DecisionTreeClassifier(class_weight={0:10,1:90},max_features=8,max_depth=10)
```

In [117…
```
evaluation(dt,x_train1,y_train,x_test1,y_test)
```

```
-----------------------------------------------
For training data confusion matrix can be given as
[[31751  1768]
 [   13  2889]]
              precision    recall  f1-score   support

         0.0       1.00      0.95      0.97     33519
         1.0       0.62      1.00      0.76      2902

    accuracy                           0.95     36421
   macro avg       0.81      0.97      0.87     36421
weighted avg       0.97      0.95      0.96     36421

accuracy score of training data is:95.10996403173993
F1 score of training data is 86.8552800948172
Precision score of training data is 62.03564526519219
Recall score of training data is 99.55203308063405
-----------------------------------------------
 for test data confusion matrix given as
[[7909  402]
 [  20  775]]
              precision    recall  f1-score   support

         0.0       1.00      0.95      0.97      8311
         1.0       0.66      0.97      0.79       795

    accuracy                           0.95      9106
   macro avg       0.83      0.96      0.88      9106
weighted avg       0.97      0.95      0.96      9106

accuracy score of test data is:95.3656929497035
F1 score of test data is 88.00094175601275
Precision score of test data is 65.84536958368734
Recall score of test data is 97.48427672955975
-----------------------------------------------
RMSE is 0.21527440745003823
ROC_AUC 0.9632365683427813
Precision Cross Validation Score 0.637999004770063
Recall Cross Validation Score 0.940366193839397
Accuracy cross validation score 0.9471181672525637
```

In [ ]:
```python
# To check which algorithm provide best results with respect to  hyperparameter so we use optuna
```

In [121…
```python
import optuna
```

In [127…
```python
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
```

In [128…
```python
def objective(trial):
    # Choose the algorithm to tune
    classifier_name = trial.suggest_categorical('classifier', ['SVM', 'RandomForest', 'GradientBoosting','Logis

    if classifier_name == 'SVM':
        # SVM hyperparameters
        c = trial.suggest_float('C', 0.1, 100, log=True)
        kernel = trial.suggest_categorical('kernel', ['linear', 'rbf', 'poly', 'sigmoid'])
        gamma = trial.suggest_categorical('gamma', ['scale', 'auto'])
        weight_0=trial.suggest_int('weight_0',1,10)
        weight_1=trial.suggest_int('weight_1',65,95)
        model = SVC(C=c, kernel=kernel, gamma=gamma,class_weight={0:weight_0,1:weight_1}, random_state=42)

    elif classifier_name == 'RandomForest':
        # Random Forest hyperparameters
        n_estimators = trial.suggest_int('n_estimators', 50, 300)
        max_depth = trial.suggest_int('max_depth', 3, 20)
        min_samples_split = trial.suggest_int('min_samples_split', 2, 10)
        min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 10)
        bootstrap = trial.suggest_categorical('bootstrap', [True, False])
        weight_0=trial.suggest_int('weight_0',1,10)
        weight_1=trial.suggest_int('weight_1',65,95)

        model = RandomForestClassifier(
            n_estimators=n_estimators,
            max_depth=max_depth,
            min_samples_split=min_samples_split,
            min_samples_leaf=min_samples_leaf,
            class_weight={0:weight_0,1:weight_1},
            bootstrap=bootstrap,
            random_state=42
        )

    elif classifier_name == 'GradientBoosting':
        # Gradient Boosting hyperparameters
```

```python
        n_estimators = trial.suggest_int('n_estimators', 50, 300)
        learning_rate = trial.suggest_float('learning_rate', 0.01, 0.3, log=True)
        max_depth = trial.suggest_int('max_depth', 3, 20)
        min_samples_split = trial.suggest_int('min_samples_split', 2, 10)
        min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 10)


        model = GradientBoostingClassifier(
            n_estimators=n_estimators,
            learning_rate=learning_rate,
            max_depth=max_depth,
            min_samples_split=min_samples_split,
            min_samples_leaf=min_samples_leaf,

            random_state=42
        )
    elif classifier_name == 'LogisticRegression':
        l1_ratio= trial.suggest_int('l1_ratio',0,1)
        weight_0=trial.suggest_int('weight_0',1,10)
        weight_1=trial.suggest_int('weight_1',65,95)

        model= LogisticRegression(
            l1_ratio=l1_ratio,
            class_weight={0:weight_0,1:weight_1},
            random_state=42
        )
    # Perform cross-validation and return the mean accuracy
    score = cross_val_score(model, x_train1, y_train, cv=3, scoring='precision').mean()
    return score
```

In [129…

```python
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=15)
```

```
[I 2025-08-09 15:51:13,195] A new study created in memory with name: no-name-c0856d86-6836-4385-bd18-16bfeda86ae
3
[I 2025-08-09 15:51:14,276] Trial 0 finished with value: 0.5909889136642517 and parameters: {'classifier': 'Logi
sticRegression', 'l1_ratio': 0, 'weight_0': 5, 'weight_1': 75}. Best is trial 0 with value: 0.5909889136642517.
[I 2025-08-09 15:51:21,671] Trial 1 finished with value: 0.6269499153282411 and parameters: {'classifier': 'Rand
omForest', 'n_estimators': 123, 'max_depth': 13, 'min_samples_split': 8, 'min_samples_leaf': 10, 'bootstrap': Tr
ue, 'weight_0': 2, 'weight_1': 73}. Best is trial 1 with value: 0.6269499153282411.
[I 2025-08-09 15:51:35,196] Trial 2 finished with value: 0.7245251290345288 and parameters: {'classifier': 'Rand
omForest', 'n_estimators': 182, 'max_depth': 18, 'min_samples_split': 8, 'min_samples_leaf': 7, 'bootstrap': Tru
e, 'weight_0': 8, 'weight_1': 67}. Best is trial 2 with value: 0.7245251290345288.
[I 2025-08-09 15:51:49,756] Trial 3 finished with value: 0.6441574192676424 and parameters: {'classifier': 'Rand
omForest', 'n_estimators': 235, 'max_depth': 17, 'min_samples_split': 9, 'min_samples_leaf': 8, 'bootstrap': Tru
e, 'weight_0': 5, 'weight_1': 87}. Best is trial 2 with value: 0.7245251290345288.
[I 2025-08-09 15:51:50,813] Trial 4 finished with value: 0.5617973882314241 and parameters: {'classifier': 'Logi
sticRegression', 'l1_ratio': 1, 'weight_0': 3, 'weight_1': 66}. Best is trial 2 with value: 0.7245251290345288.
[I 2025-08-09 15:52:10,372] Trial 5 finished with value: 0.645544308004654 and parameters: {'classifier': 'SVM',
'C': 20.979989208994954, 'kernel': 'rbf', 'gamma': 'auto', 'weight_0': 2, 'weight_1': 94}. Best is trial 2 with
value: 0.7245251290345288.
[I 2025-08-09 15:53:00,765] Trial 6 finished with value: 0.5036751623305317 and parameters: {'classifier': 'SVM'
, 'C': 5.879603981398609, 'kernel': 'linear', 'gamma': 'scale', 'weight_0': 2, 'weight_1': 75}. Best is trial 2
with value: 0.7245251290345288.
[I 2025-08-09 15:53:12,609] Trial 7 finished with value: 0.6660034231964169 and parameters: {'classifier': 'Rand
omForest', 'n_estimators': 188, 'max_depth': 14, 'min_samples_split': 7, 'min_samples_leaf': 8, 'bootstrap': Tru
e, 'weight_0': 7, 'weight_1': 73}. Best is trial 2 with value: 0.7245251290345288.
[I 2025-08-09 15:54:47,153] Trial 8 finished with value: 0.9068246404753645 and parameters: {'classifier': 'Grad
ientBoosting', 'n_estimators': 232, 'learning_rate': 0.03423164438979198, 'max_depth': 20, 'min_samples_split':
6, 'min_samples_leaf': 1}. Best is trial 8 with value: 0.9068246404753645.
[I 2025-08-09 16:08:09,171] Trial 9 finished with value: 0.6098656241316057 and parameters: {'classifier': 'SVM'
, 'C': 83.8374425566565, 'kernel': 'linear', 'gamma': 'scale', 'weight_0': 7, 'weight_1': 67}. Best is trial 8 w
ith value: 0.9068246404753645.
[I 2025-08-09 16:10:11,849] Trial 10 finished with value: 0.9719294456313511 and parameters: {'classifier': 'Gra
dientBoosting', 'n_estimators': 299, 'learning_rate': 0.034362661841781265, 'max_depth': 4, 'min_samples_split':
3, 'min_samples_leaf': 1}. Best is trial 10 with value: 0.9719294456313511.
[I 2025-08-09 16:12:13,438] Trial 11 finished with value: 0.9735695794595004 and parameters: {'classifier': 'Gra
dientBoosting', 'n_estimators': 299, 'learning_rate': 0.03385209973783653, 'max_depth': 4, 'min_samples_split':
2, 'min_samples_leaf': 1}. Best is trial 11 with value: 0.9735695794595004.
[I 2025-08-09 16:13:45,202] Trial 12 finished with value: 0.9853344355720989 and parameters: {'classifier': 'Gra
dientBoosting', 'n_estimators': 299, 'learning_rate': 0.035315513911943895, 'max_depth': 3, 'min_samples_split':
2, 'min_samples_leaf': 1}. Best is trial 12 with value: 0.9853344355720989.
[I 2025-08-09 16:15:19,451] Trial 13 finished with value: 0.9300192932883663 and parameters: {'classifier': 'Gra
dientBoosting', 'n_estimators': 300, 'learning_rate': 0.16265035592951058, 'max_depth': 3, 'min_samples_split':
2, 'min_samples_leaf': 3}. Best is trial 12 with value: 0.9853344355720989.
[I 2025-08-09 16:15:40,396] Trial 14 finished with value: 0.0 and parameters: {'classifier': 'GradientBoosting',
'n_estimators': 53, 'learning_rate': 0.010374717767639584, 'max_depth': 7, 'min_samples_split': 4, 'min_samples_
leaf': 4}. Best is trial 12 with value: 0.9853344355720989.
```

In [147…

```python
print(f"best trial accuracy:{study.best_trial.value}")
print(f"best Hyperparameters:{study.best_trial.params}")
```

```
best trial accuracy:0.9853344355720989
best Hyperparameters:{'classifier': 'GradientBoosting', 'n_estimators': 299, 'learning_rate': 0.035315513911943895, 'max_depth': 3, 'min_samples_split': 2, 'min_samples_leaf': 1}
```

In [ ]:
```
# The best result provided by Gradient Boosting Classifier with following Hyperparameters
```

In [148…
```
gb=GradientBoostingClassifier(n_estimators=299,
                              learning_rate=0.035315513911943895,
                              max_depth=3,
                              min_samples_split=2,
                              min_samples_leaf=1)
```

In [149…
```
evaluation(gb,x_train1,y_train,x_test1,y_test)
```

```
--------------------------------------------------
For training data confusion matrix can be given as
[[33513     6]
 [  671  2231]]
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     33519
         1.0       1.00      0.77      0.87      2902

    accuracy                           0.98     36421
   macro avg       0.99      0.88      0.93     36421
weighted avg       0.98      0.98      0.98     36421

accuracy score of training data is:98.1411822849455
F1 score of training data is 92.91313754769095
Precision score of training data is 99.73178363880196
Recall score of training data is 76.87801516195726
--------------------------------------------------
 for test data confusion matrix given as
[[8307    4]
 [ 186  609]]
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99      8311
         1.0       0.99      0.77      0.87       795

    accuracy                           0.98      9106
   macro avg       0.99      0.88      0.93      9106
weighted avg       0.98      0.98      0.98      9106

accuracy score of test data is:97.91346365034043
F1 score of test data is 92.68749932375408
Precision score of test data is 99.34747145187602
Recall score of test data is 76.60377358490567
--------------------------------------------------
RMSE is 0.1444484804232833
ROC_AUC 0.8827782229961202
Precision Cross Validation Score 0.9910303990504404
Recall Cross Validation Score 0.7584408570241558
Accuracy cross validation score 0.9801762439803753
```

In [134…
```
from sklearn.ensemble import AdaBoostClassifier
abc=AdaBoostClassifier()
```

In [135…
```
evaluation(abc,x_train1,y_train,x_test1,y_test)
```

```
-------------------------------------------------
For training data confusion matrix can be given as
[[33458    61]
 [  624  2278]]
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     33519
         1.0       0.97      0.78      0.87      2902

    accuracy                           0.98     36421
   macro avg       0.98      0.89      0.93     36421
weighted avg       0.98      0.98      0.98     36421

accuracy score of training data is:98.1192169352846
F1 score of training data is 92.95833828786525
Precision score of training data is 97.39204788371099
Recall score of training data is 78.49758787043419
-------------------------------------------------
 for test data confusion matrix given as
[[8300   11]
 [ 160  635]]
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99      8311
         1.0       0.98      0.80      0.88       795

    accuracy                           0.98      9106
   macro avg       0.98      0.90      0.94      9106
weighted avg       0.98      0.98      0.98      9106

accuracy score of test data is:98.12211728530639
F1 score of test data is 93.55681180432285
Precision score of test data is 98.29721362229103
Recall score of test data is 79.87421383647799
-------------------------------------------------
RMSE is 0.13703586080634544
ROC_AUC 0.8987092956292676
Precision Cross Validation Score 0.9551914972663624
Recall Cross Validation Score 0.7822203098106713
Accuracy cross validation score 0.9797095051743238
```

In [138…

```python
from xgboost import XGBClassifier
xgb=XGBClassifier()
```

In [139…

```python
evaluation(xgb,x_train1,y_train,x_test1,y_test)
```

```
-------------------------------------------------
For training data confusion matrix can be given as
[[33519     0]
 [   27  2875]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     33519
         1.0       1.00      0.99      1.00      2902

    accuracy                           1.00     36421
   macro avg       1.00      1.00      1.00     36421
weighted avg       1.00      1.00      1.00     36421

accuracy score of training data is:99.92586694489442
F1 score of training data is 99.74618497131534
Precision score of training data is 100.0
Recall score of training data is 99.0696071674707
-------------------------------------------------
 for test data confusion matrix given as
[[8272   39]
 [ 146  649]]
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99      8311
         1.0       0.94      0.82      0.88       795

    accuracy                           0.98      9106
   macro avg       0.96      0.91      0.93      9106
weighted avg       0.98      0.98      0.98      9106

accuracy score of test data is:97.96837250164727
F1 score of test data is 93.209711256435
Precision score of test data is 94.3313953488372
Recall score of test data is 81.63522012578616
-------------------------------------------------
RMSE is 0.1425351710404395
ROC_AUC 0.9058298125769516
Precision Cross Validation Score 0.9054450981922996
Recall Cross Validation Score 0.8032399548934656
Accuracy cross validation score 0.9776227622750969
```

In [140…
```
# To check feature importance
```

In [141…
```
feature_importance=rf.feature_importances_
```

In [142…
```
print(feature_importance)
```

```
[2.87449774e-03 3.79524816e-03 7.47288137e-04 1.05275463e-03
 1.59091296e-03 1.56278004e-03 1.24592251e-04 9.43987679e-04
 1.60570073e-05 2.41952473e-03 5.48669020e-04 1.39351338e-03
 7.47021018e-04 3.03871879e-04 3.11387359e-04 2.00298908e-03
 1.59720593e-04 1.02873614e-03 2.30315256e-03 3.33204404e-04
 3.65586450e-03 1.95835504e-02 2.56560592e-02 2.55197555e-02
 7.62770447e-02 2.57358947e-02 2.61067286e-02 3.36952406e-01
 2.69576114e-01 1.51839117e-01 4.59859632e-03 7.25197559e-03
 2.98698549e-03]
```

In [143…
```
importance_column=pd.DataFrame({'Feature':x_train1.columns, 'Importance':feature_importance})
```

In [144…
```
importance_column
```

Out[144...

	Feature	Importance
0	gender	0.002874
1	owns_house	0.003795
2	Cleaning staff_ohe	0.000747
3	Cooking staff_ohe	0.001053
4	Core staff_ohe	0.001591
5	Drivers_ohe	0.001563
6	HR staff_ohe	0.000125
7	High skill tech staff_ohe	0.000944
8	IT staff_ohe	0.000016
9	Laborers_ohe	0.002420
10	Low-skill Laborers_ohe	0.000549
11	Managers_ohe	0.001394
12	Medicine staff _ohe	0.000747
13	Private service staff_ohe	0.000304
14	Realty agents_ohe	0.000311
15	Sales staff_ohe	0.002003
16	Secretaries_ohe	0.000160
17	Security staff_ohe	0.001029
18	Unknown_ohe	0.002303
19	Waiters/barmen staff_ohe	0.000333
20	owns_car_imputed	0.003656
21	Age	0.019584
22	net_yearly_income	0.025656
23	credit_limit	0.025520
24	credit_limit_used(%)	0.076277
25	median_no_of_days_employed	0.025736
26	median_yearly_debt_payments	0.026107
27	median_credit_score	0.336952
28	prev_defaults	0.269576
29	default_in_last_6months	0.151839
30	no_of_children_imputed	0.004599
31	total_family_members_imputed	0.007252
32	migrant_worker_imputed	0.002987

In [145...

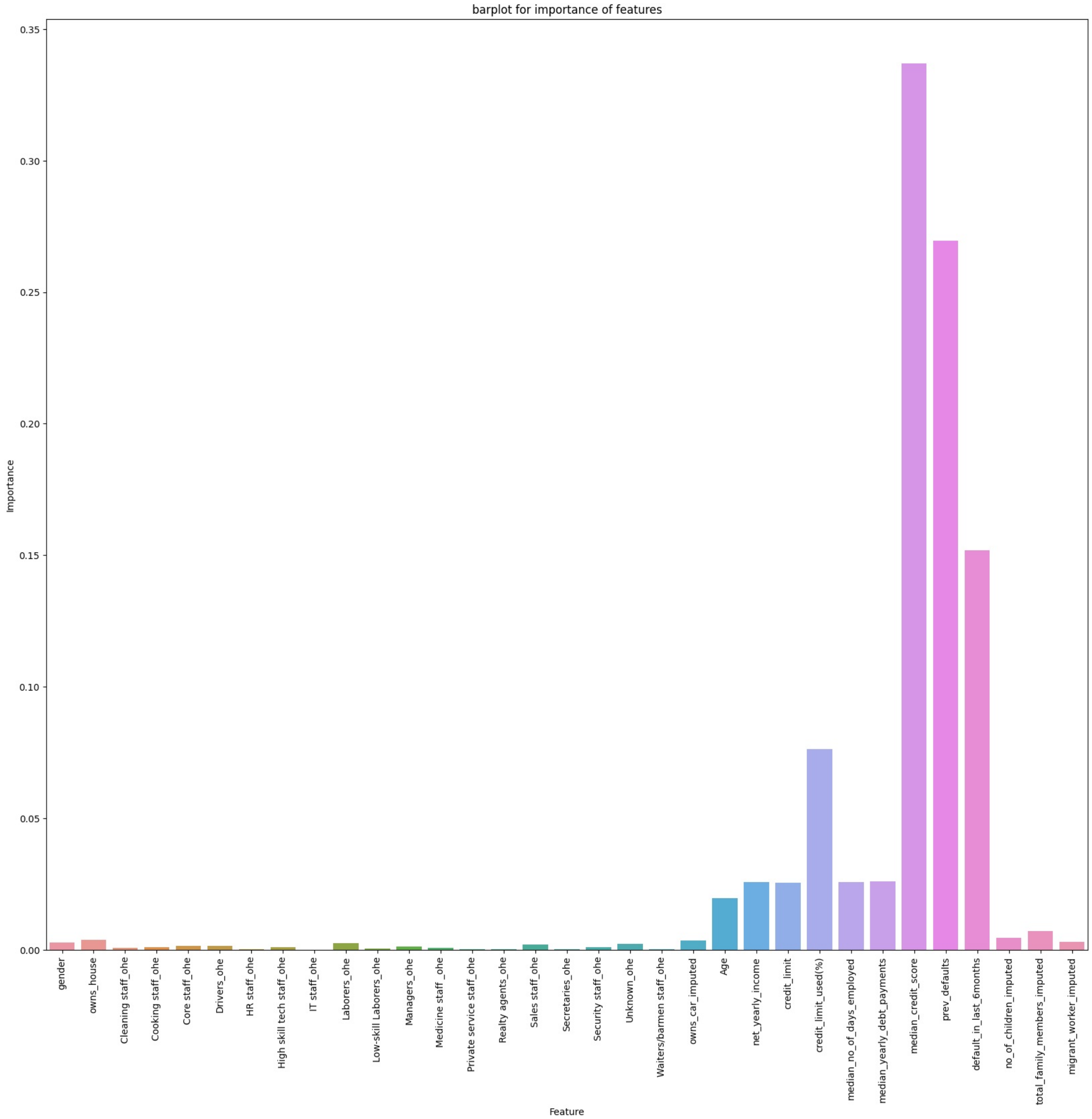
```
importance_column['Importance'].sum()
```

Out[145... 1.0

Importance of each feature

In [146...

```
plt.figure(figsize=(20,18))
sns.barplot(data=importance_column,x='Feature',y='Importance')
plt.xticks(rotation=90)
plt.title(f"barplot for importance of features")
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js