In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
sns.set_theme(color_codes=True)
pd.set_option('display.max_columns', None)
```

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

```
C:\Users\Michael\AppData\Local\Temp\ipykernel_29436\608555978.py:1: DtypeWarn
ing: Columns (1,7,8,16,17,18,19,20,35) have mixed types. Specify dtype option
on import or set low_memory=False.
  df = pd.read_csv('Train_Dataset.csv')
```

Out[2]:

ıg	Client_Contact_Work_Tag	Type_Organization	Score_Source_1	Score_Source_2	Score_Source_3
es	Yes	Self-employed	0.568066	0.478787	NaN
es	Yes	Government	0.563360	0.215068	NaN
es	Yes	Self-employed	NaN	0.552795	0.329655
es	Yes	XNA	NaN	0.135182	0.631355
es	Yes	Business Entity Type 3	0.508199	0.301182	0.355639

Data Preprocessing Part 1

In [3]:
```
# Drop identifier column
df.drop(columns = ['ID', 'ID_Days'], inplace=True)
df.head()
```

Out[3]:

latch_Tag	Client_Contact_Work_Tag	Type_Organization	Score_Source_1	Score_Source_2	Score_Sc
Yes	Yes	Self-employed	0.568066	0.478787	
Yes	Yes	Government	0.563360	0.215068	
Yes	Yes	Self-employed	NaN	0.552795	0
Yes	Yes	XNA	NaN	0.135182	0
Yes	Yes	Business Entity Type 3	0.508199	0.301182	0

In [4]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[4]:
```
Client_Income                 1516
Credit_Amount                 6816
Loan_Annuity                 16680
Accompany_Client                 7
Client_Income_Type               8
Client_Education                 5
Client_Marital_Status            4
Client_Gender                    3
Loan_Contract_Type               2
Client_Housing_Type              6
Population_Region_Relative     164
Age_Days                     22583
Employed_Days                13220
Registration_Days            19254
Client_Occupation               18
Client_Permanent_Match_Tag       2
Client_Contact_Work_Tag          2
Type_Organization               58
Score_Source_3                1430
dtype: int64
```

In [5]:
```
#Change numerical column datatypes from object to int and remove string data
df['Client_Income'] = df['Client_Income'].str.replace('[^0-9.]', '', regex=Tru
df['Client_Income'] = pd.to_numeric(df['Client_Income'], errors='coerce')

df['Credit_Amount'] = df['Credit_Amount'].str.replace('[^0-9.]', '', regex=Tru
df['Credit_Amount'] = pd.to_numeric(df['Credit_Amount'], errors='coerce')

df['Loan_Annuity'] = df['Loan_Annuity'].str.replace('[^0-9.]', '', regex=True)
df['Loan_Annuity'] = pd.to_numeric(df['Loan_Annuity'], errors='coerce')

df['Population_Region_Relative'] = df['Population_Region_Relative'].str.replac
df['Population_Region_Relative'] = pd.to_numeric(df['Population_Region_Relativ

df['Age_Days'] = df['Age_Days'].str.replace('[^0-9.]', '', regex=True)
df['Age_Days'] = pd.to_numeric(df['Age_Days'], errors='coerce')

df['Employed_Days'] = df['Employed_Days'].str.replace('[^0-9.]', '', regex=Tru
df['Employed_Days'] = pd.to_numeric(df['Employed_Days'], errors='coerce')

df['Registration_Days'] = df['Registration_Days'].str.replace('[^0-9.]', '', r
df['Registration_Days'] = pd.to_numeric(df['Registration_Days'], errors='coerc

df['Score_Source_3'] = df['Score_Source_3'].str.replace('[^0-9.]', '', regex=T
df['Score_Source_3'] = pd.to_numeric(df['Score_Source_3'], errors='coerce')
```

In [6]:
```
df.head()
```

Out[6]:

latch_Tag	Client_Contact_Work_Tag	Type_Organization	Score_Source_1	Score_Source_2	Score_Sc
Yes	Yes	Self-employed	0.568066	0.478787	
Yes	Yes	Government	0.563360	0.215068	
Yes	Yes	Self-employed	NaN	0.552795	
Yes	Yes	XNA	NaN	0.135182	
Yes	Yes	Business Entity Type 3	0.508199	0.301182	

In [7]:
```
df.dtypes
```

Out[7]:
```
Client_Income                float64
Car_Owned                    float64
Bike_Owned                   float64
Active_Loan                  float64
House_Own                    float64
Child_Count                  float64
Credit_Amount                float64
Loan_Annuity                 float64
Accompany_Client              object
Client_Income_Type            object
Client_Education              object
Client_Marital_Status         object
Client_Gender                 object
Loan_Contract_Type            object
Client_Housing_Type           object
Population_Region_Relative   float64
Age_Days                     float64
Employed_Days                float64
Registration_Days            float64
Own_House_Age                float64
Mobile_Tag                     int64
Homephone_Tag                  int64
Workphone_Working              int64
Client_Occupation             object
Client_Family_Members        float64
Cleint_City_Rating           float64
Application_Process_Day      float64
Application_Process_Hour     float64
Client_Permanent_Match_Tag    object
Client_Contact_Work_Tag       object
Type_Organization             object
Score_Source_1               float64
Score_Source_2               float64
Score_Source_3               float64
Social_Circle_Default        float64
Phone_Change                 float64
Credit_Bureau                float64
Default                        int64
dtype: object
```

In [8]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[8]:
```
Accompany_Client              7
Client_Income_Type            8
Client_Education              5
Client_Marital_Status         4
Client_Gender                 3
Loan_Contract_Type            2
Client_Housing_Type           6
Client_Occupation            18
Client_Permanent_Match_Tag    2
Client_Contact_Work_Tag       2
Type_Organization            58
dtype: int64
```

Segment Client Occupation into smaller Unique Value

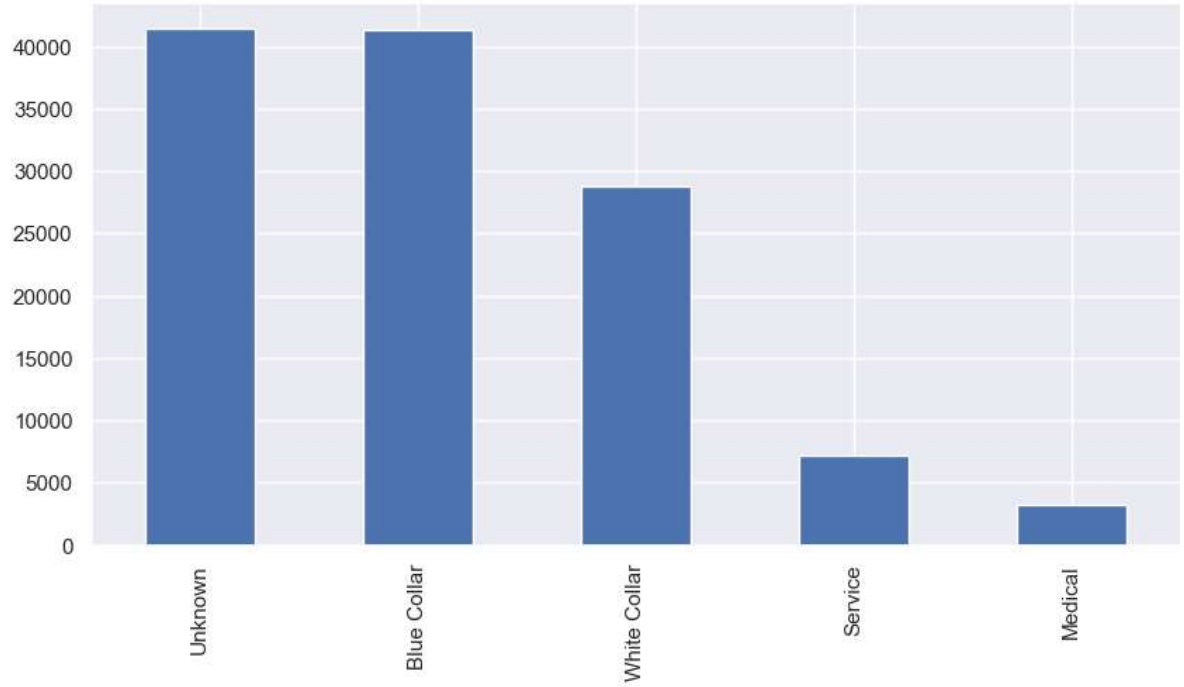
In [9]:
```
df.Client_Occupation.unique()
```

Out[9]:
```
array(['Sales', nan, 'Realty agents', 'Laborers', 'Core', 'Drivers',
       'Managers', 'Accountants', 'High skill tech', 'Cleaning', 'HR',
       'Waiters/barmen', 'Low-skill Laborers', 'Medicine', 'Cooking',
       'Private service', 'Security', 'IT', 'Secretaries'], dtype=object)
```

In [10]:
```
def map_occupation(category):
    if pd.isna(category):
        return 'Unknown'
    elif category in ['Sales', 'Realty agents', 'Managers', 'Accountants', 'Hi
        return 'White Collar'
    elif category in ['Laborers', 'Core', 'Drivers', 'Cleaning', 'Low-skill La
        return 'Blue Collar'
    elif category in ['HR', 'Waiters/barmen', 'Cooking', 'Private service', 'S
        return 'Service'
    elif category == 'Medicine':
        return 'Medical'
    else:
        return 'Other'

# Apply the function to the 'Client_Occupation' column and create a new 'Occup
df['Client_Occupation'] = df['Client_Occupation'].apply(map_occupation)
```

In [11]:
```
plt.figure(figsize=(10,5))
df['Client_Occupation'].value_counts().plot(kind='bar')
```

Out[11]:
```
<AxesSubplot:>
```



Segment Client Occupation into smaller Unique Value

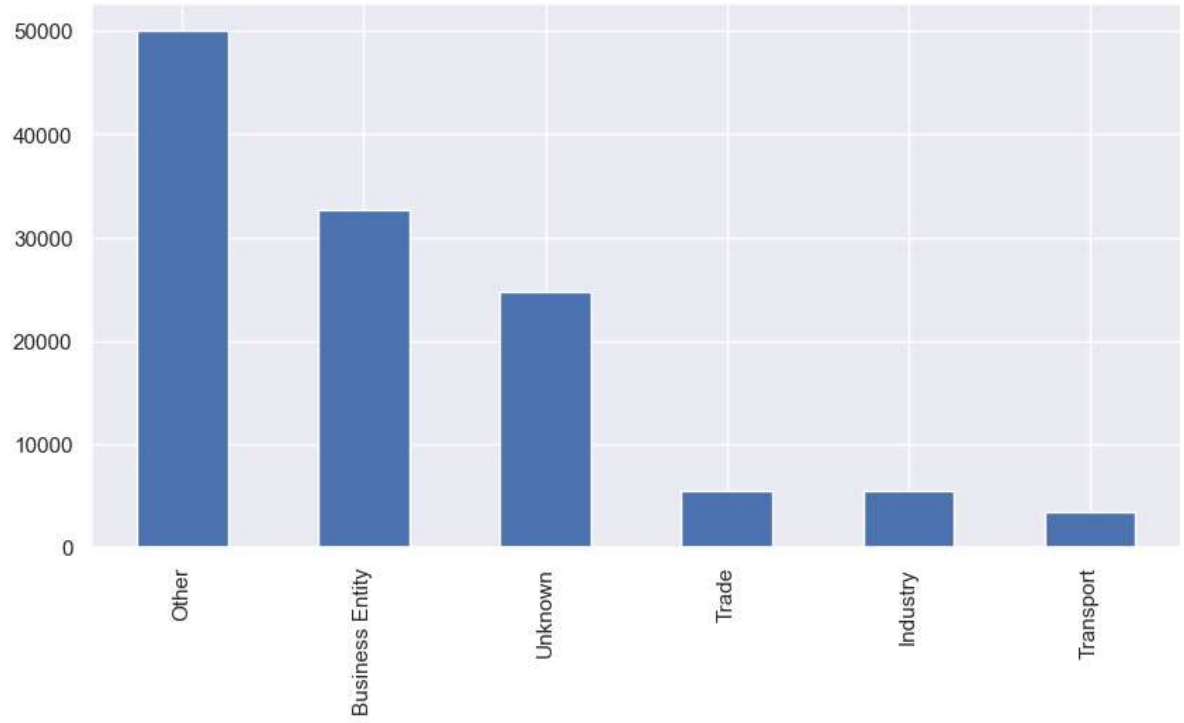
In [12]:
```
df.Type_Organization.unique()
```

Out[12]:
```
array(['Self-employed', 'Government', 'XNA', 'Business Entity Type 3',
       'Other', nan, 'Industry: type 3', 'Business Entity Type 2',
       'Business Entity Type 1', 'Transport: type 4', 'Construction',
       'Kindergarten', 'Trade: type 3', 'Industry: type 2',
       'Trade: type 7', 'Trade: type 2', 'Agriculture', 'Military',
       'Medicine', 'Housing', 'Industry: type 1', 'Industry: type 11',
       'Bank', 'School', 'Industry: type 9', 'Postal', 'University',
       'Transport: type 2', 'Restaurant', 'Electricity', 'Police',
       'Industry: type 4', 'Security Ministries', 'Services',
       'Transport: type 3', 'Mobile', 'Hotel', 'Security',
       'Industry: type 7', 'Advertising', 'Cleaning', 'Realtor',
       'Trade: type 6', 'Culture', 'Industry: type 5', 'Telecom',
       'Trade: type 1', 'Industry: type 12', 'Industry: type 8',
       'Insurance', 'Emergency', 'Legal Services', 'Industry: type 10',
       'Trade: type 4', 'Industry: type 6', 'Transport: type 1',
       'Industry: type 13', 'Religion', 'Trade: type 5'], dtype=object)
```

In [13]:
```
# Define a function to map original values to new categories
def map_organization(category):
    if pd.isna(category):
        return 'Unknown'
    elif category == 'XNA':
        return 'Unknown'  # Replace 'XNA' with 'Unknown'
    elif 'Business Entity Type' in category:
        return 'Business Entity'  # Group all 'Business Entity Type' together
    elif 'Industry: type' in category:
        return 'Industry'  # Group all 'Industry: type' together
    elif 'Trade: type' in category:
        return 'Trade'  # Group all 'Trade: type' together
    elif 'Transport: type' in category:
        return 'Transport'  # Group all 'Transport: type' together
    else:
        return 'Other'

# Apply the function to the 'Type_Organization' column
df['Type_Organization'] = df['Type_Organization'].apply(map_organization)
```

In [14]:
```
plt.figure(figsize=(10,5))
df['Type_Organization'].value_counts().plot(kind='bar')
```

Out[14]: <AxesSubplot:>



In [15]:
```
# Replace 1 with 'yes' and 0 with 'no' in the categorical column
df['Car_Owned'] = df['Car_Owned'].replace({0: 'no', 1: 'yes'})
df['Bike_Owned'] = df['Bike_Owned'].replace({0: 'no', 1: 'yes'})
df['Active_Loan'] = df['Active_Loan'].replace({0: 'no', 1: 'yes'})
df['House_Own'] = df['House_Own'].replace({0: 'no', 1: 'yes'})
df['Mobile_Tag'] = df['Mobile_Tag'].replace({0: 'no', 1: 'yes'})
df['Homephone_Tag'] = df['Homephone_Tag'].replace({0: 'no', 1: 'yes'})
df['Workphone_Working'] = df['Workphone_Working'].replace({0: 'no', 1: 'yes'})
df['Default'] = df['Default'].replace({0: 'no', 1: 'yes'})
```

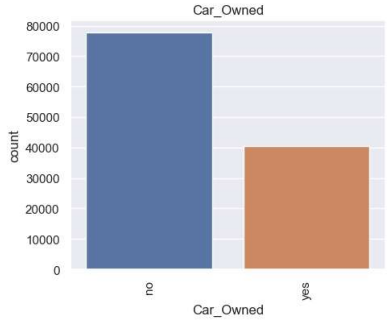
In [16]:
```
df.head()
```

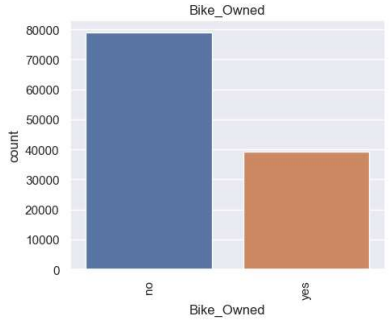
Out[16]:

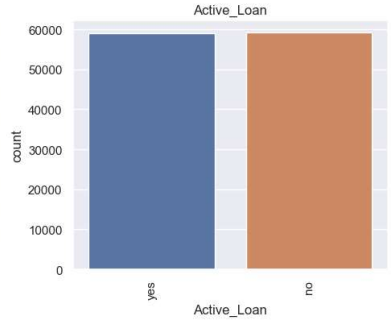
latch_Tag	Client_Contact_Work_Tag	Type_Organization	Score_Source_1	Score_Source_2	Score_Sc
Yes	Yes	Other	0.568066	0.478787	
Yes	Yes	Other	0.563360	0.215068	
Yes	Yes	Other	NaN	0.552795	
Yes	Yes	Unknown	NaN	0.135182	
Yes	Yes	Business Entity	0.508199	0.301182	

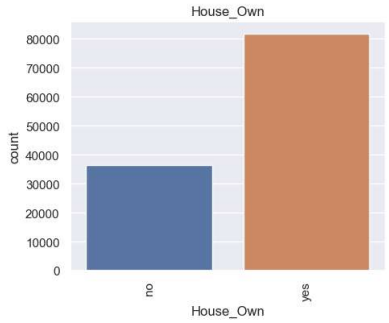
Exploratory Data Analysis

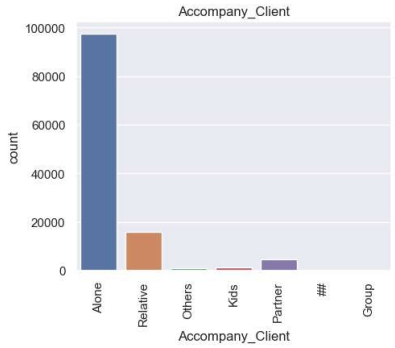
In [17]:
```
# Get the names of all columns with data type 'object' (categorical columns) ex
cat_vars = df.select_dtypes(include='object').columns.tolist()

# Create a figure with subplots
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a countplot for the top 6 values of each categorical variable using Se
for i, var in enumerate(cat_vars):
    top_values = df[var].value_counts().index
    filtered_df = df[df[var].isin(top_values)]
    sns.countplot(x=var, data=filtered_df, ax=axs[i])
    axs[i].set_title(var)
    axs[i].tick_params(axis='x', rotation=90)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

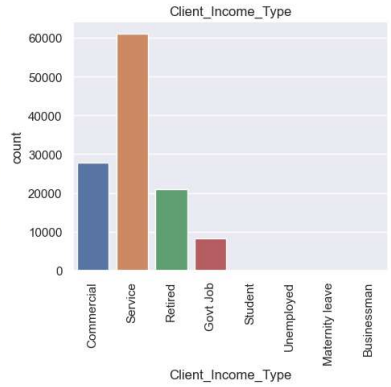

Car_Owned
count
no
yes
Car_Owned

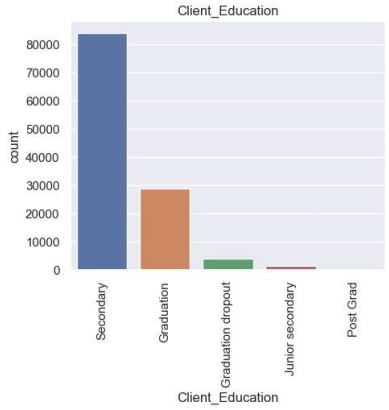

Bike_Owned
count
no
yes
Bike_Owned

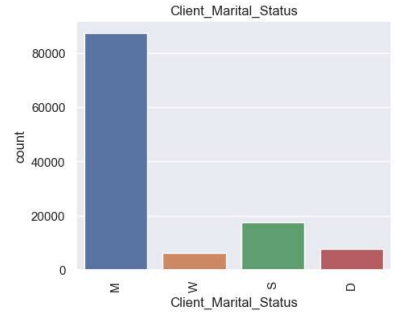

Active_Loan
count
yes
no
Active_Loan

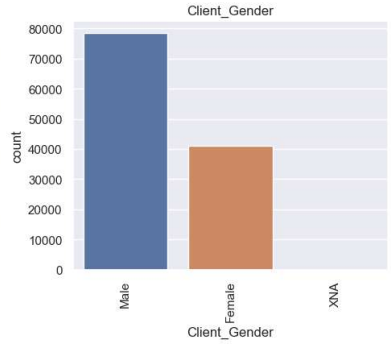

House_Own
count
no
yes
House_Own

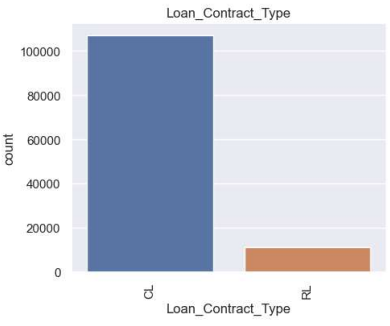

Accompany_Client
count
Alone
Relative
Others
Kids
Partner
##
Group
Accompany_Client

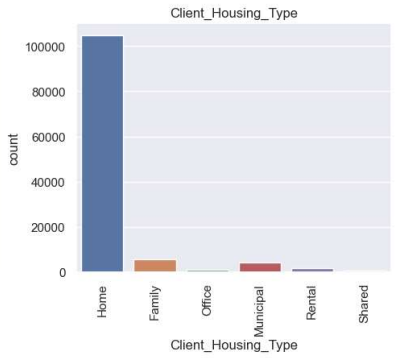

Client_Income_Type
count
Commercial
Service
Retired
Govt Job
Student
Unemployed
Maternity leave
Businessman
Client_Income_Type

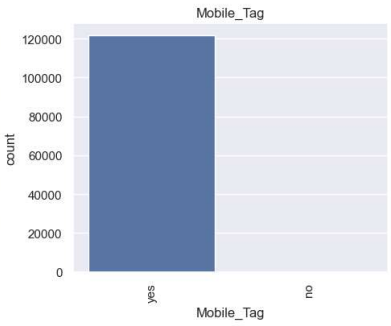

Client_Education
count
Secondary
Graduation
Graduation dropout
Junior secondary
Post Grad
Client_Education

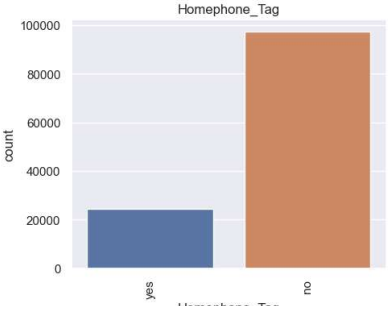

Client_Marital_Status
count
M
W
S
D
Client_Marital_Status

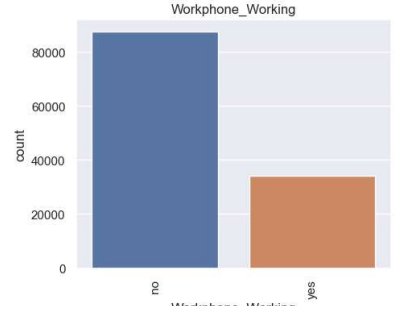

Client_Gender
count
Male
Female
XNA
Client_Gender

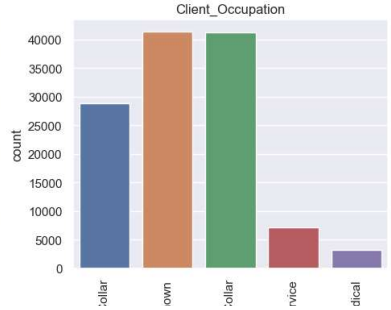

Loan_Contract_Type
count
CL
RL
Loan_Contract_Type

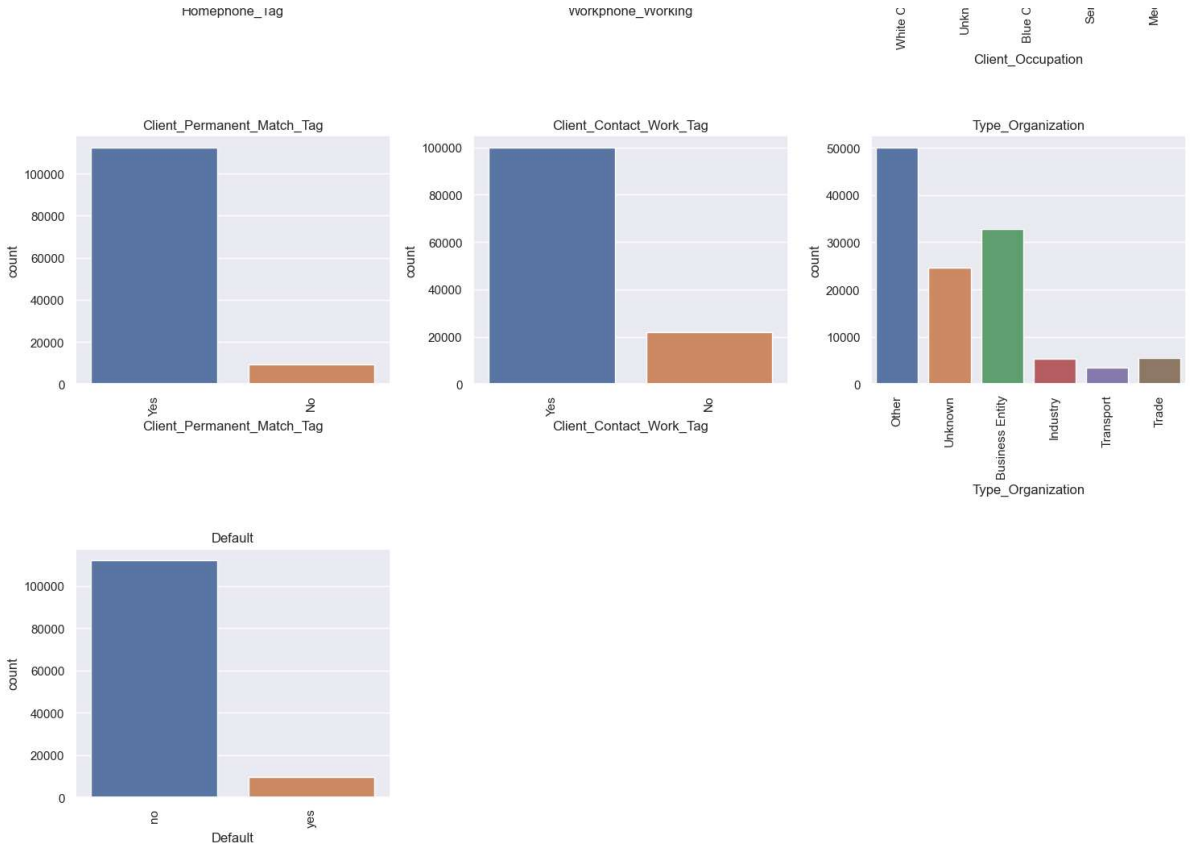

Client_Housing_Type
count
Home
Family
Office
Municipal
Rental
Shared
Client_Housing_Type


Mobile_Tag
count
yes
no
Mobile_Tag

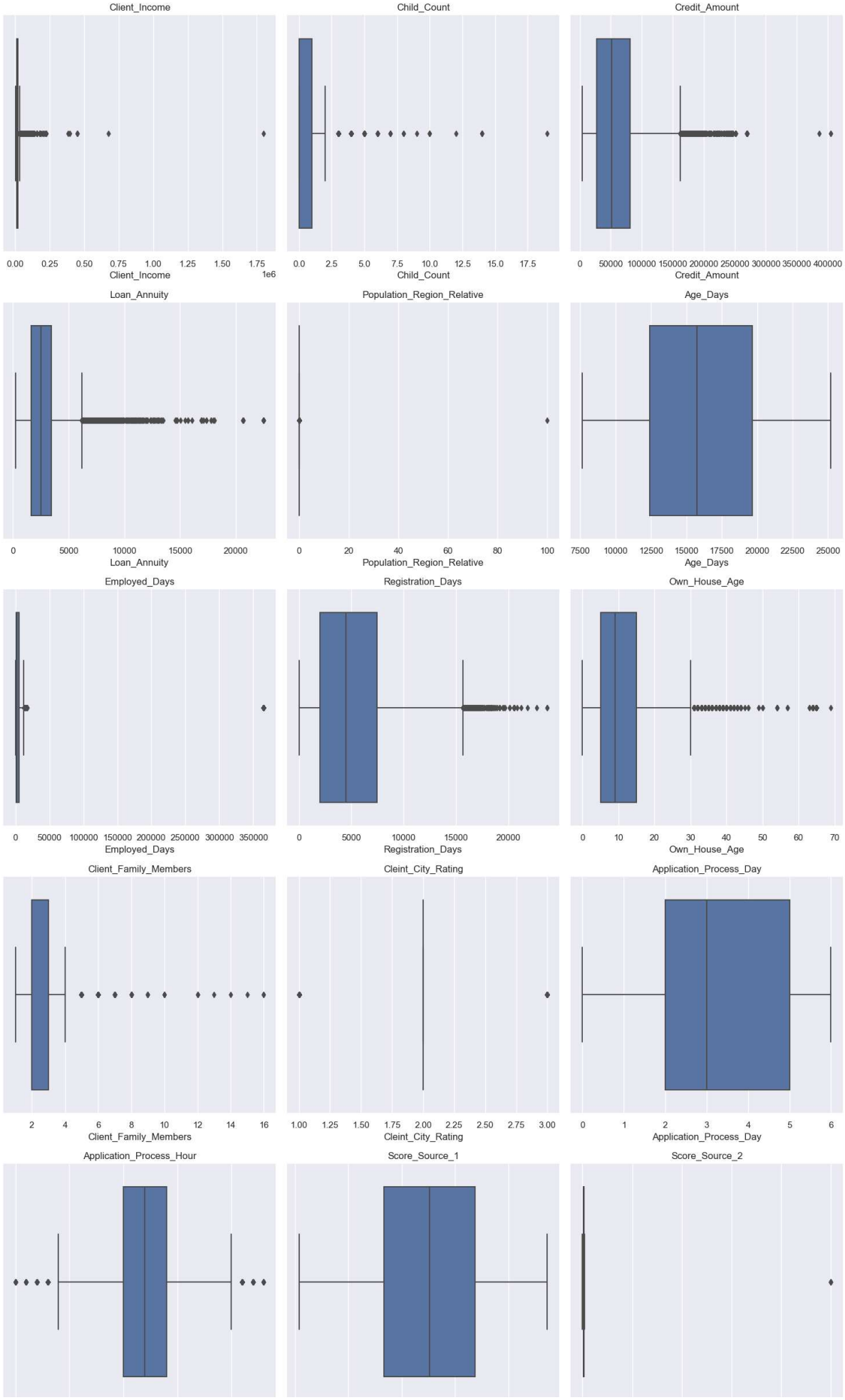

Homephone_Tag
count
yes
no

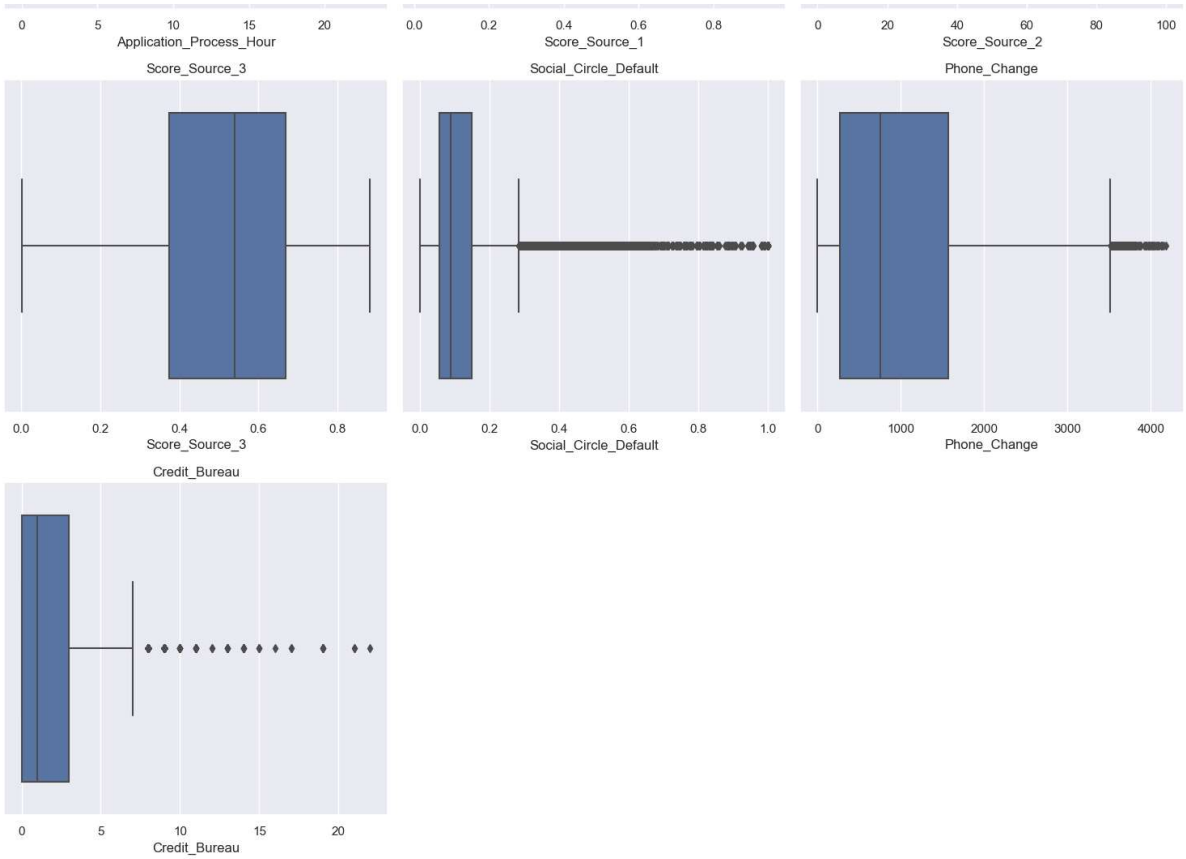

Workphone_Working
count
no
yes


Client_Occupation
count

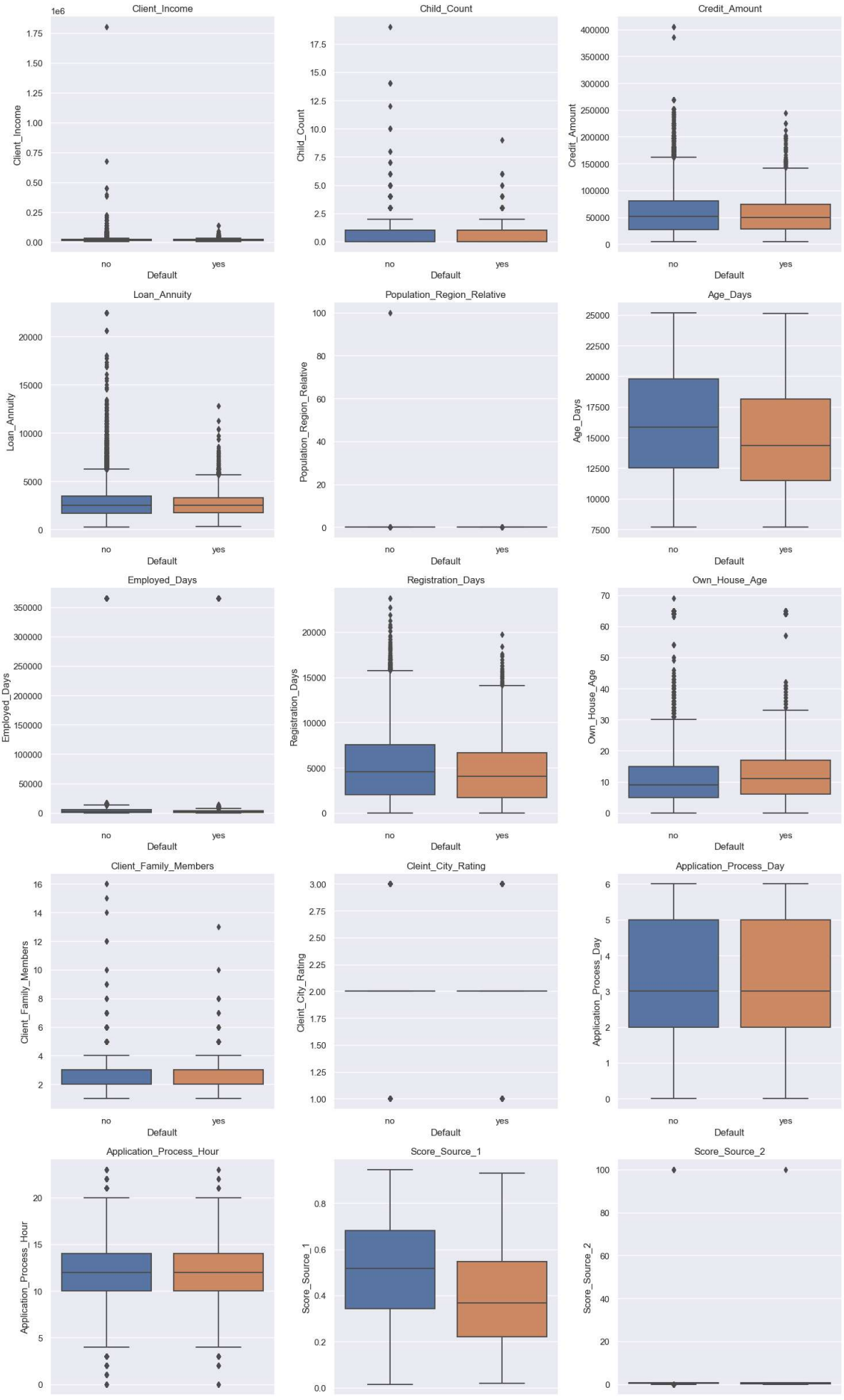

Homephone_Tag
Workphone_Working
White C
Unkn
Blue C
Client_Occupation
Client_Permanent_Match_Tag
count
Yes
No
Client_Permanent_Match_Tag
Client_Contact_Work_Tag
count
Yes
No
Client_Contact_Work_Tag
Type_Organization
count
Other
Unknown
Business Entity
Industry
Transport
Trade
Type_Organization
Default
count
no
yes
Default

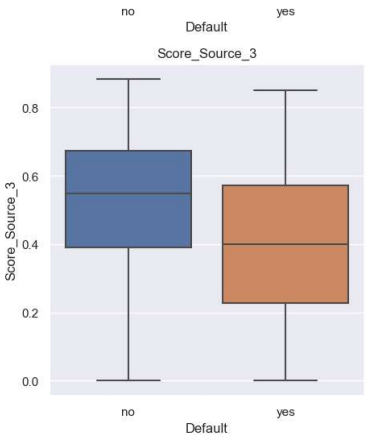
In [18]:
```
# Get the names of all columns with data type 'int' or 'float', excluding the s
num_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(num_vars)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each numerical variable using Seaborn
for i, var in enumerate(num_vars):
    sns.boxplot(x=df[var], ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

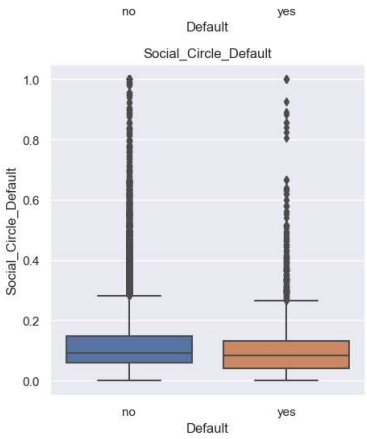

Client_Income
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75
Client_Income
1e6
Child_Count
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Child_Count
Credit_Amount
0 50000 100000 150000 200000 250000 300000 350000 400000
Credit_Amount
Loan_Annuity
0 5000 10000 15000 20000
Loan_Annuity
Population_Region_Relative
0 20 40 60 80 100
Population_Region_Relative
Age_Days
7500 10000 12500 15000 17500 20000 22500 25000
Age_Days
Employed_Days
0 50000 100000 150000 200000 250000 300000 350000
Employed_Days
Registration_Days
0 5000 10000 15000 20000
Registration_Days
Own_House_Age
0 10 20 30 40 50 60 70
Own_House_Age
Client_Family_Members
2 4 6 8 10 12 14 16
Client_Family_Members
Cleint_City_Rating
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00
Cleint_City_Rating
Application_Process_Day
0 1 2 3 4 5 6
Application_Process_Day
Application_Process_Hour
Score_Source_1
Score_Source_2

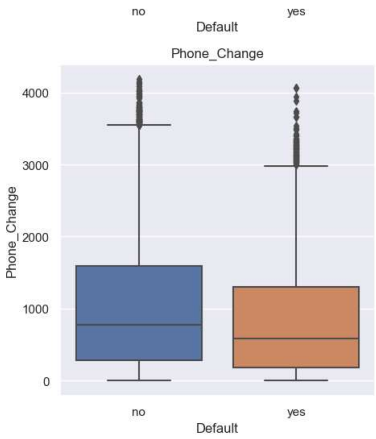

Application_Process_Hour
Score_Source_1
Score_Source_2
Score_Source_3
Social_Circle_Default
Phone_Change
Credit_Bureau

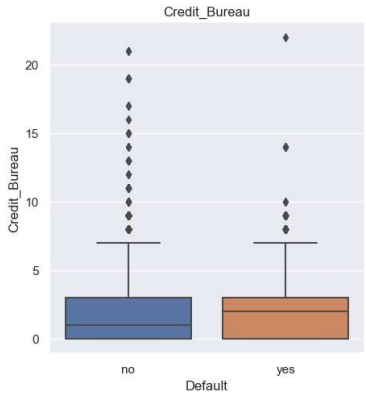
In [19]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the s
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each integer variable using Seaborn with hue='attritio
for i, var in enumerate(int_vars):
    sns.boxplot(y=var, x='Default', data=df, ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


Client_Income
1e6
Client_Income
Default
Child_Count
Child_Count
Default
Credit_Amount
Credit_Amount
Default
Loan_Annuity
Loan_Annuity
Default
Population_Region_Relative
Population_Region_Relative
Default
Age_Days
Age_Days
Default
Employed_Days
Employed_Days
Default
Registration_Days
Registration_Days
Default
Own_House_Age
Own_House_Age
Default
Client_Family_Members
Client_Family_Members
Default
Cleint_City_Rating
Cleint_City_Rating
Default
Application_Process_Day
Application_Process_Day
Default
Application_Process_Hour
Application_Process_Hour
Score_Source_1
Score_Source_1
Score_Source_2
Score_Source_2
no
yes

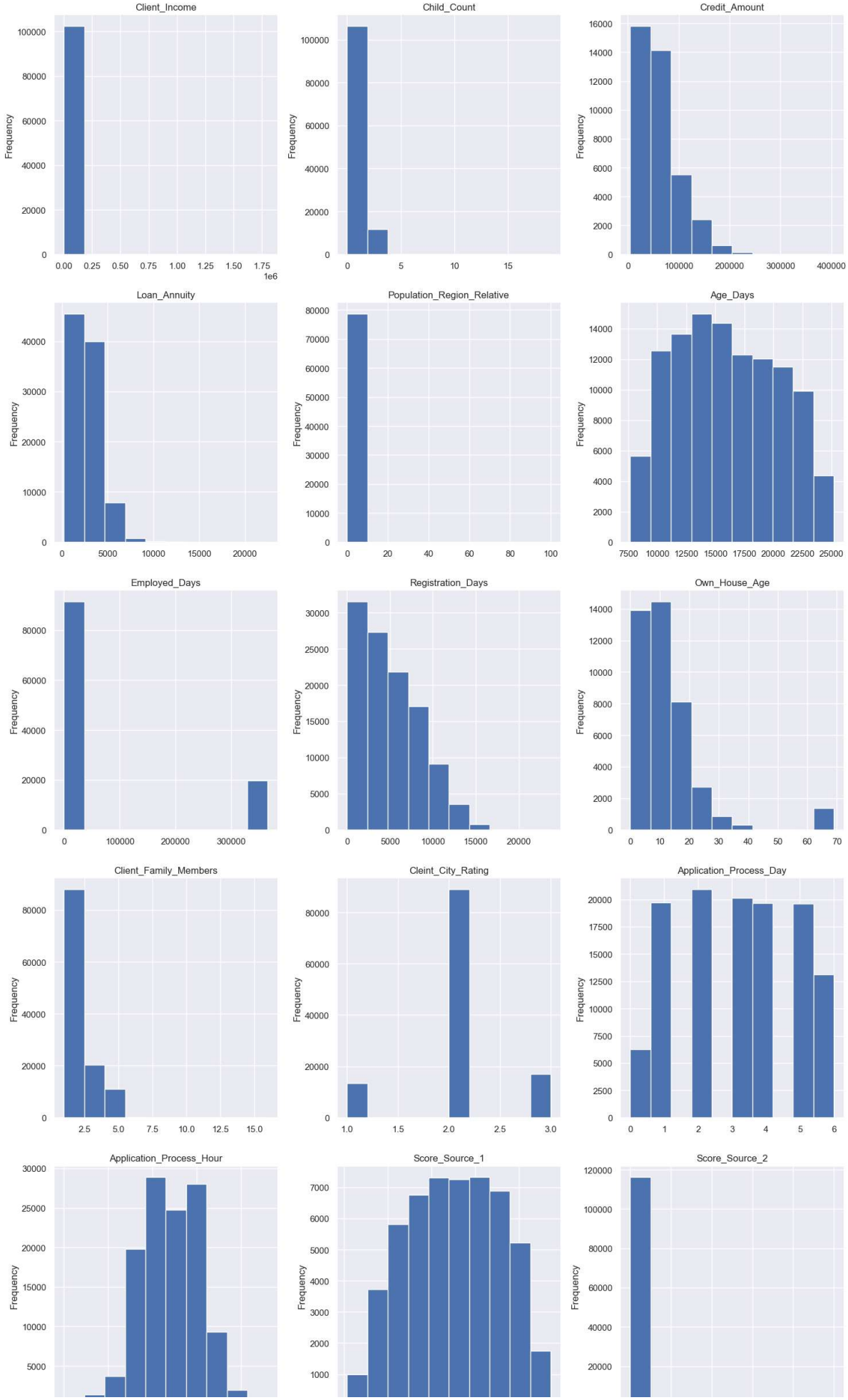

Score_Source_3
Score_Source_3
Default
no
yes
0.0
0.2
0.4
0.6
0.8


Social_Circle_Default
Social_Circle_Default
Default
no
yes
0.0
0.2
0.4
0.6
0.8
1.0


Phone_Change
Phone_Change
Default
no
yes
0
1000
2000
3000
4000


Credit_Bureau
Credit_Bureau
Default
no
yes
0
5
10
15
20

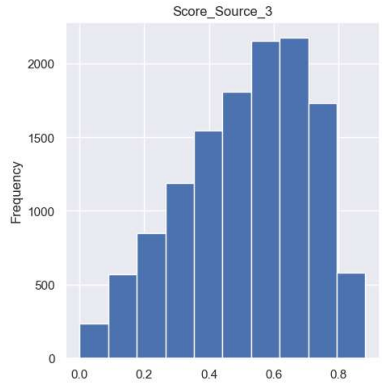
In [20]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the s
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable
for i, var in enumerate(int_vars):
    df[var].plot.hist(ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

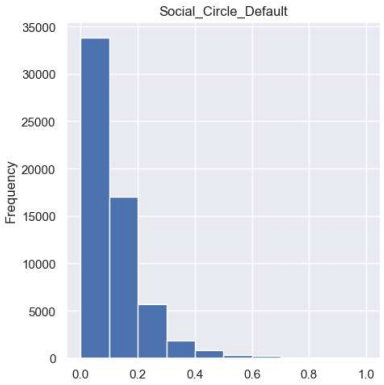

Client_Income
Frequency
1e6
Child_Count
Credit_Amount
Loan_Annuity
Population_Region_Relative
Age_Days
Employed_Days
Registration_Days
Own_House_Age
Client_Family_Members
Cleint_City_Rating
Application_Process_Day
Application_Process_Hour
Score_Source_1
Score_Source_2

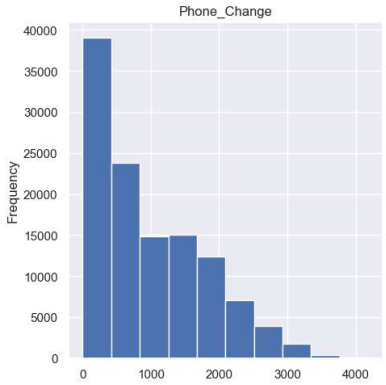

0
0
5
10
15
20

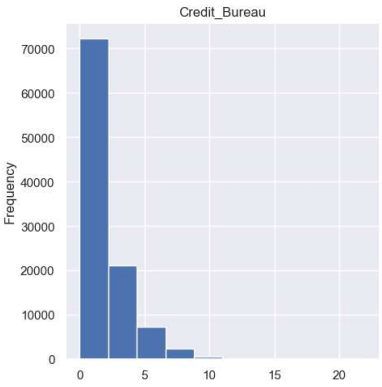

0
0.0
0.2
0.4
0.6
0.8


0
0
20
40
60
80
100

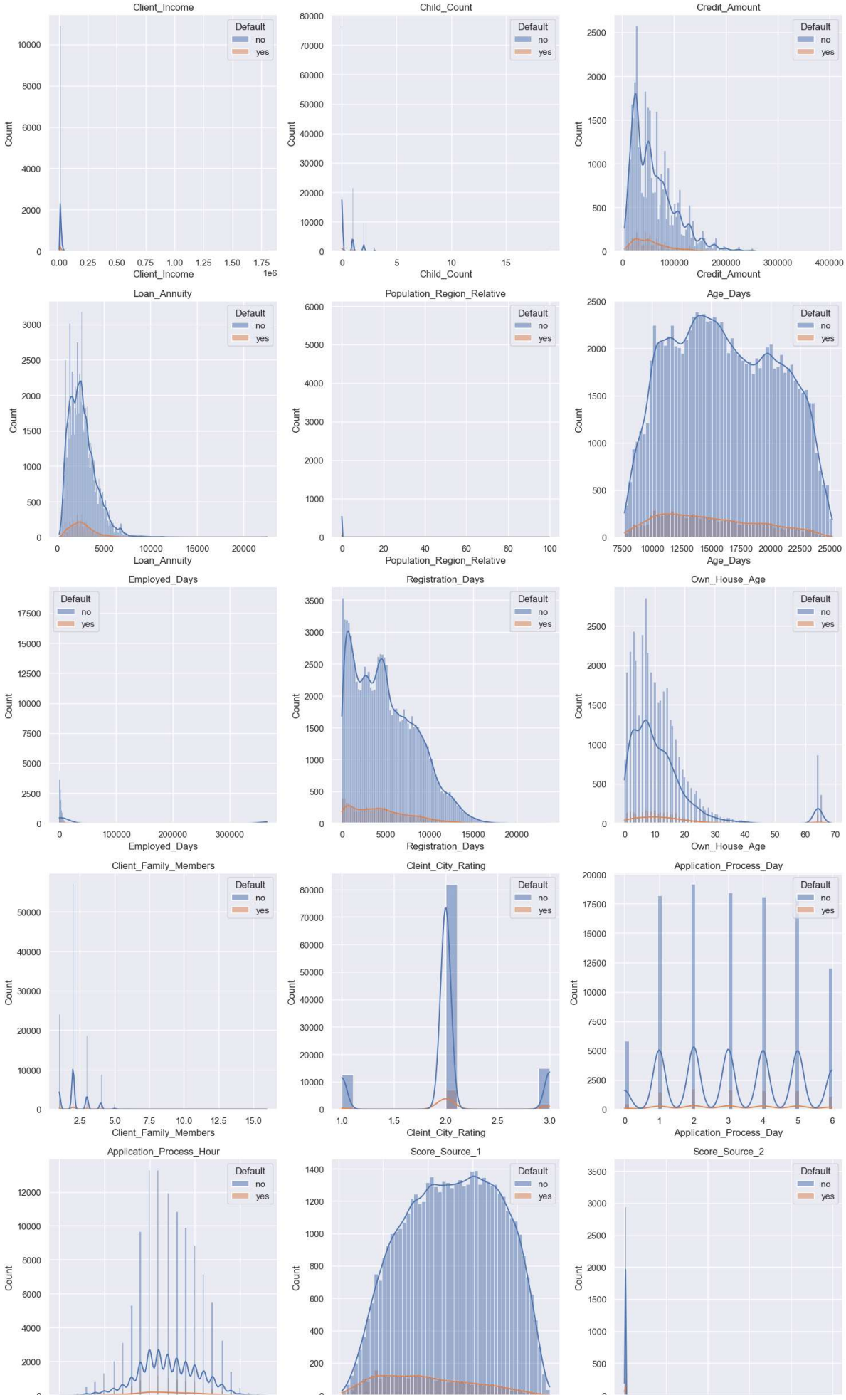

Score_Source_3
Frequency
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8

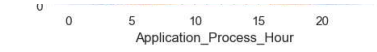

Social_Circle_Default
Frequency
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

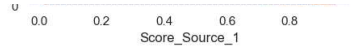

Phone_Change
Frequency
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000


Credit_Bureau
Frequency
70000
60000
50000
40000
30000
20000
10000
0
0
5
10
15
20

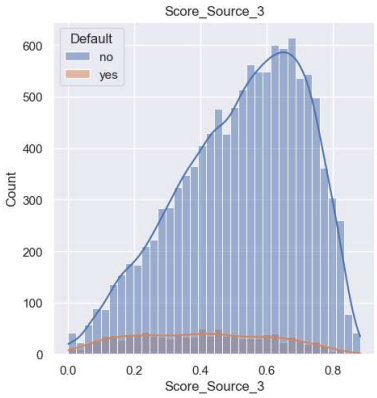
In [21]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the s
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable with hue='Attrition'
for i, var in enumerate(int_vars):
    sns.histplot(data=df, x=var, hue='Default', kde=True, ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

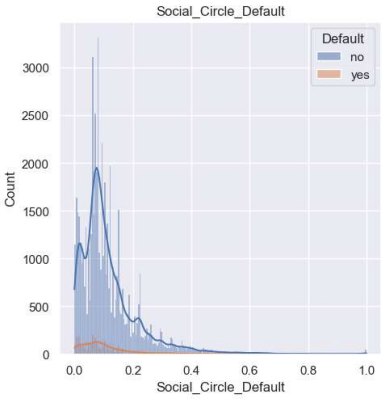

Client_Income
Child_Count
Credit_Amount
Loan_Annuity
Population_Region_Relative
Age_Days
Employed_Days
Registration_Days
Own_House_Age
Client_Family_Members
Cleint_City_Rating
Application_Process_Day
Application_Process_Hour
Score_Source_1
Score_Source_2
Default
no
yes
Count

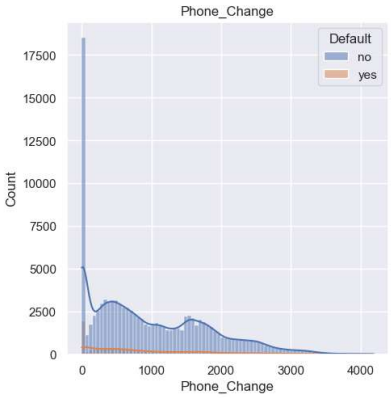

0
0
5
10
15
20
Application_Process_Hour

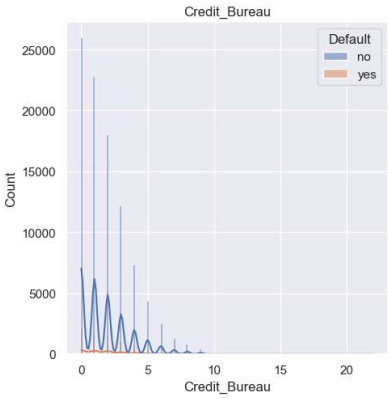

0
0.0
0.2
0.4
0.6
0.8
Score_Source_1


0
0
20
40
60
80
100
Score_Source_2

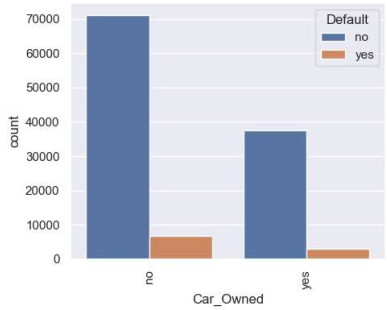

Score_Source_3
Default
no
yes
Count
0
100
200
300
400
500
600
0.0
0.2
0.4
0.6
0.8
Score_Source_3

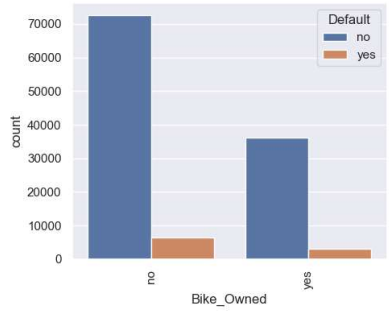

Social_Circle_Default
Default
no
yes
Count
0
500
1000
1500
2000
2500
3000
0.0
0.2
0.4
0.6
0.8
1.0
Social_Circle_Default

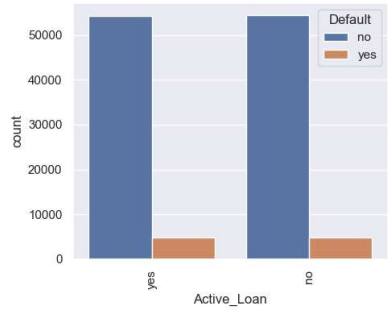

Phone_Change
Default
no
yes
Count
0
2500
5000
7500
10000
12500
15000
17500
0
1000
2000
3000
4000
Phone_Change

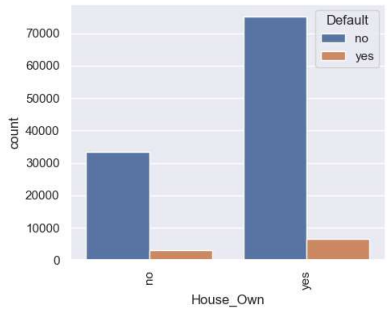

Credit_Bureau
Default
no
yes
Count
0
5000
10000
15000
20000
25000
0
5
10
15
20
Credit_Bureau

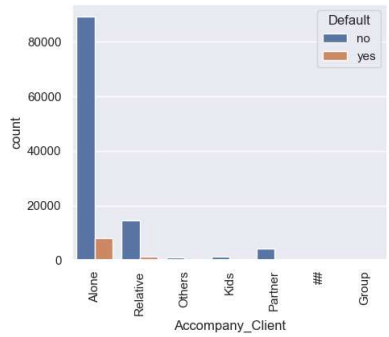
In [22]:
```
# Get the names of all columns with data type 'object' (categorical variables)
cat_vars = df.select_dtypes(include=['object']).columns.tolist()

# Exclude 'Country' from the list if it exists in cat_vars
if 'Default' in cat_vars:
    cat_vars.remove('Default')

# Create a figure with subplots, but only include the required number of subpl
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the s
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a count plot for each categorical variable
for i, var in enumerate(cat_vars):
    filtered_df = df[df[var].notnull()]  # Exclude rows with NaN values in the
    sns.countplot(x=var, hue='Default', data=filtered_df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# Remove any remaining blank subplots
for i in range(num_cols, len(axs)):
    fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show the plot
plt.show()
```

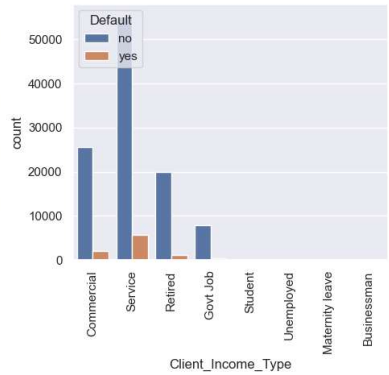

Default
no
yes
count
Car_Owned

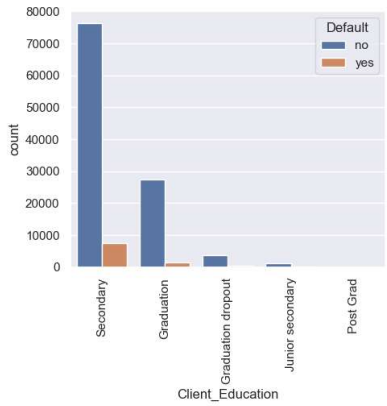

Default
no
yes
count
Bike_Owned

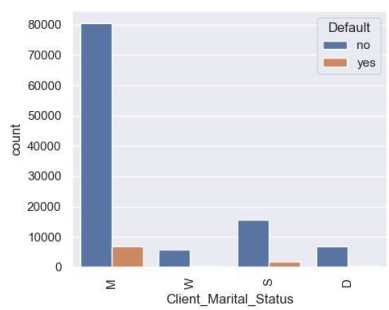

Default
no
yes
count
Active_Loan

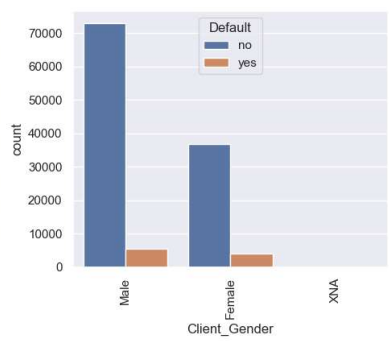

Default
no
yes
count
House_Own

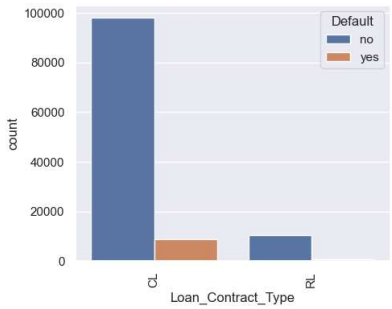

Default
no
yes
count
Alone
Relative
Others
Kids
Partner
##
Group
Accompany_Client

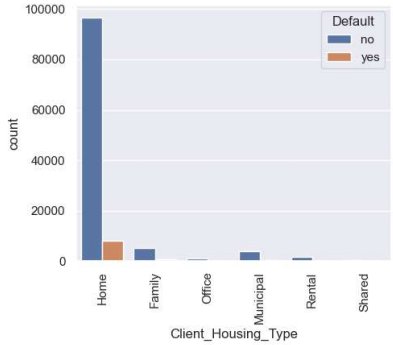

Default
no
yes
count
Commercial
Service
Retired
Govt Job
Student
Unemployed
Maternity leave
Businessman
Client_Income_Type

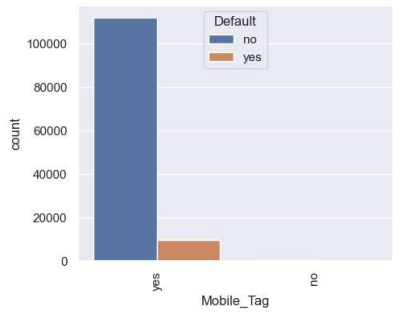

Default
no
yes
count
Secondary
Graduation
Graduation dropout
Junior secondary
Post Grad
Client_Education

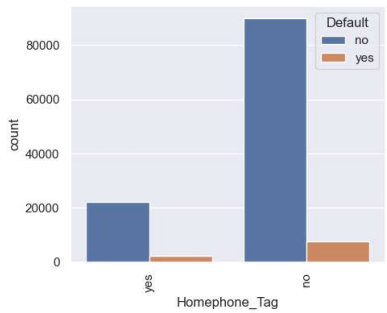

Default
no
yes
count
M
W
S
D
Client_Marital_Status

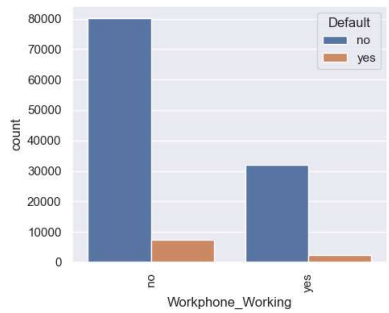

Default
no
yes
count
Male
Female
XNA
Client_Gender

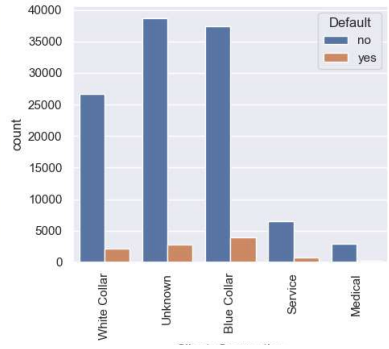

Default
no
yes
count
CL
RL
Loan_Contract_Type


Default
no
yes
count
Home
Family
Office
Municipal
Rental
Shared
Client_Housing_Type

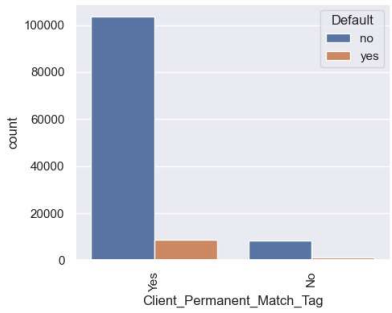

Default
no
yes
count
yes
no
Mobile_Tag

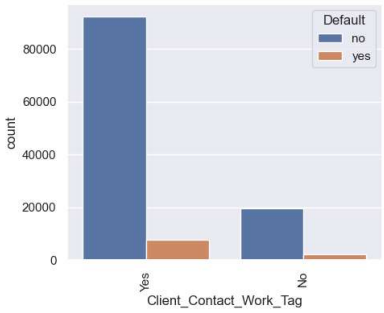

Default
no
yes
count
yes
no
Homephone_Tag

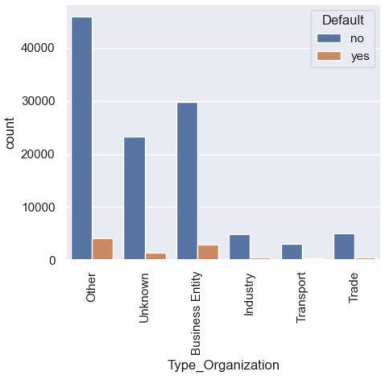

Default
no
yes
count
no
yes
Workphone_Working


Default
no
yes
count
White Collar
Unknown
Blue Collar
Service
Medical

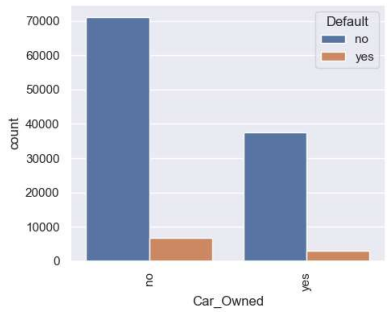
Client_Occupation

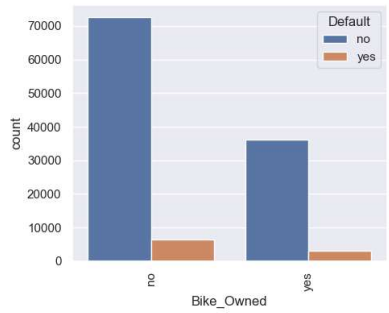


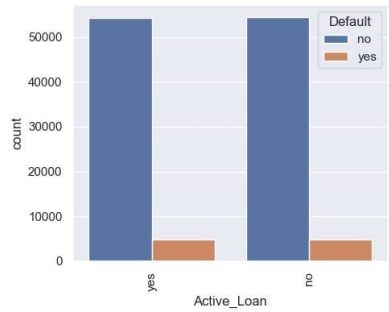


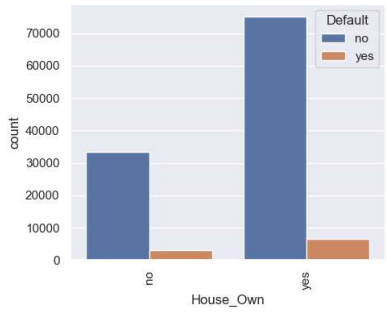


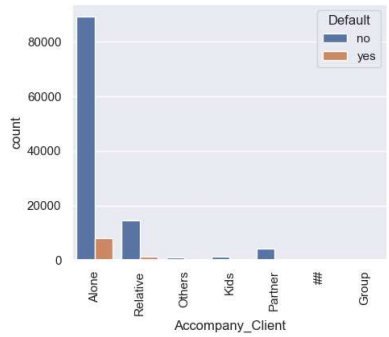
In [23]:
```
# Get the names of all columns with data type 'object' (categorical variables)
cat_vars = df.select_dtypes(include=['object']).columns.tolist()

# Exclude 'Country' and 'Employed' from the list if they exist in cat_vars
exclude_columns = ['Default']
cat_vars = [var for var in cat_vars if var not in exclude_columns]

# Create a figure with subplots, but only include the required number of subpl
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the s
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a count plot for each categorical variable
for i, var in enumerate(cat_vars):
    filtered_df = df[df[var].notnull()]  # Exclude rows with NaN values in the
    sns.countplot(x=var, hue='Default', data=filtered_df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# Remove any remaining blank subplots
for i in range(num_cols, len(axs)):
    fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show the plot
plt.show()
```

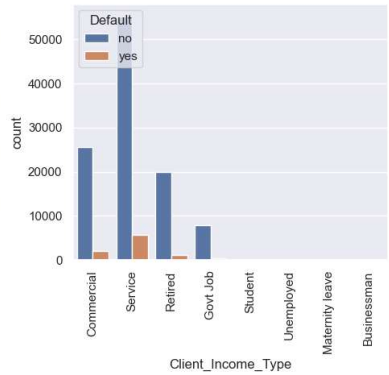

Default
no
yes
count
Car_Owned

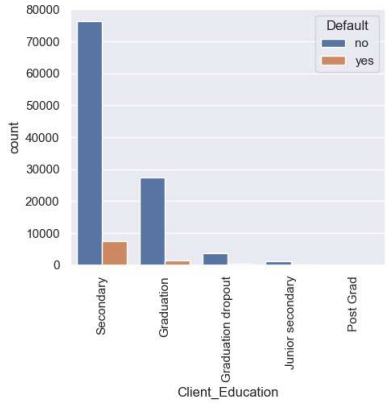

Default
no
yes
count
Bike_Owned

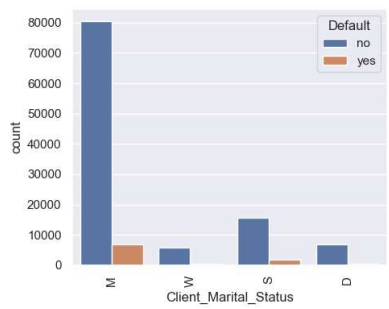

Default
no
yes
count
Active_Loan

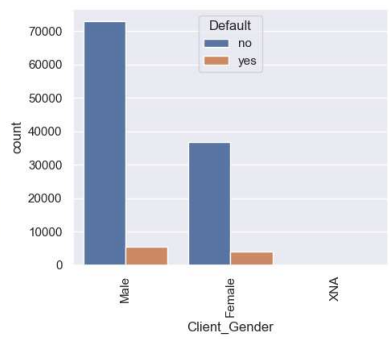

Default
no
yes
count
House_Own

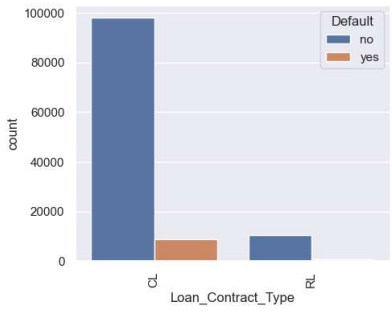

Default
no
yes
count
Alone
Relative
Others
Kids
Partner
##
Group
Accompany_Client

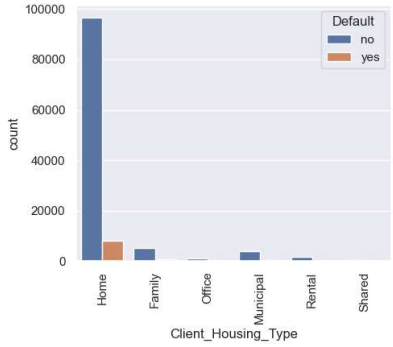

Default
no
yes
count
Commercial
Service
Retired
Govt Job
Student
Unemployed
Maternity leave
Businessman
Client_Income_Type

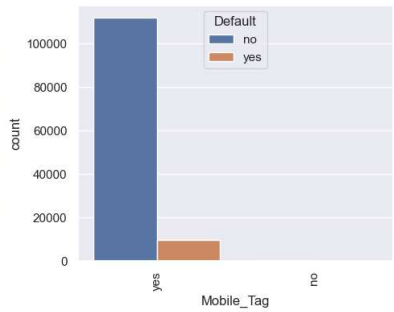

Default
no
yes
count
Secondary
Graduation
Graduation dropout
Junior secondary
Post Grad
Client_Education

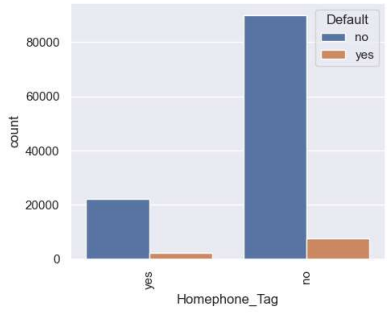

Default
no
yes
count
M
W
S
D
Client_Marital_Status

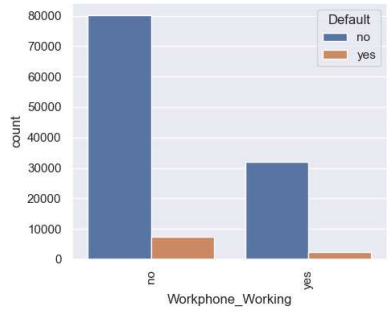

Default
no
yes
count
Male
Female
XNA
Client_Gender

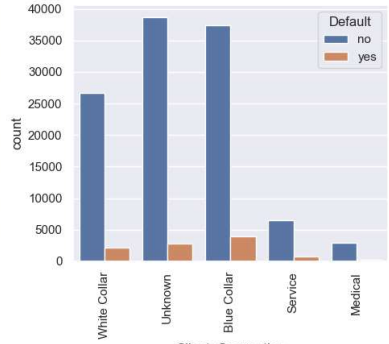

Default
no
yes
count
CL
RL
Loan_Contract_Type


Default
no
yes
count
Home
Family
Office
Municipal
Rental
Shared
Client_Housing_Type

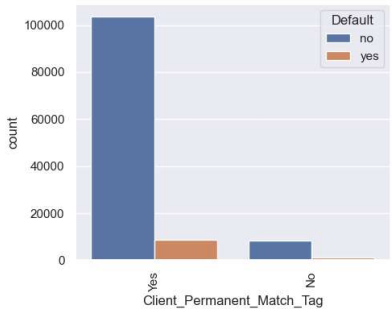

Default
no
yes
count
yes
no
Mobile_Tag

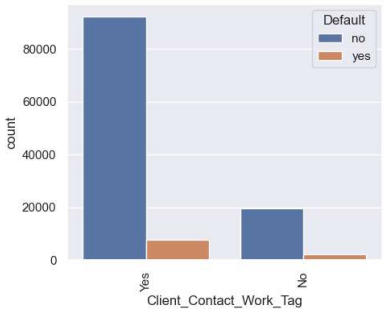

Default
no
yes
count
yes
no
Homephone_Tag

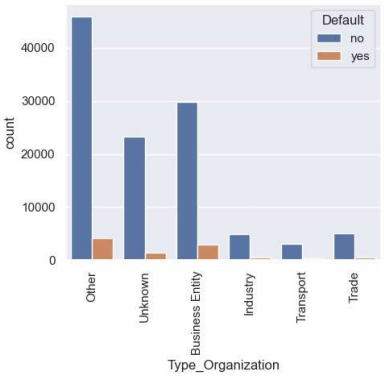

Default
no
yes
count
no
yes
Workphone_Working


Default
no
yes
count
White Collar
Unknown
Blue Collar
Service
Medical

Client_Occupation

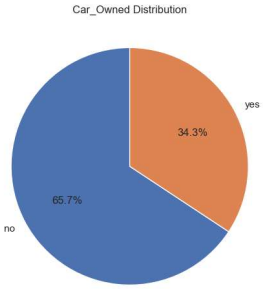


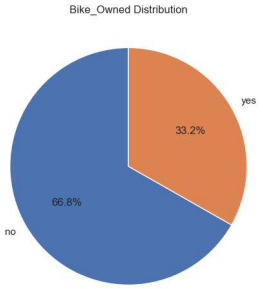


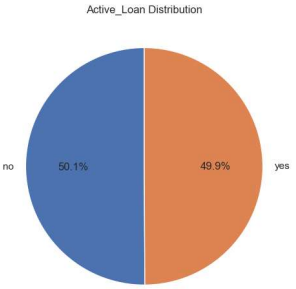


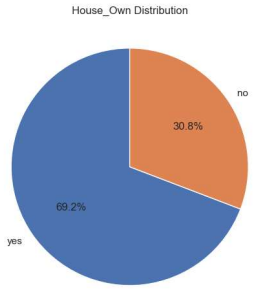
In [24]:
```
# Specify the maximum number of categories to show individually
max_categories = 5

# Filter categorical columns with 'object' data type
cat_cols = [col for col in df.columns if col != 'Country' and df[col].dtype ==

# Create a figure with subplots
num_cols = len(cat_cols)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(20, 5*num_rows))

# Flatten the axs array for easier indexing
axs = axs.flatten()

# Create a pie chart for each categorical column
for i, col in enumerate(cat_cols):
    if i < len(axs):  # Ensure we don't exceed the number of subplots
        # Count the number of occurrences for each category
        cat_counts = df[col].value_counts()

        # Group categories beyond the top max_categories as 'Other'
        if len(cat_counts) > max_categories:
            cat_counts_top = cat_counts[:max_categories]
            cat_counts_other = pd.Series(cat_counts[max_categories:].sum(), in
            cat_counts = cat_counts_top.append(cat_counts_other)

        # Create a pie chart
        axs[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', sta
        axs[i].set_title(f'{col} Distribution')

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

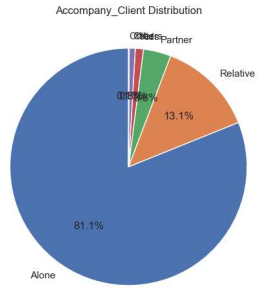
```
C:\Users\Michael\AppData\Local\Temp\ipykernel_29436\3325101321.py:25: FutureW
arning: The series.append method is deprecated and will be removed from panda
s in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
C:\Users\Michael\AppData\Local\Temp\ipykernel_29436\3325101321.py:25: FutureW
arning: The series.append method is deprecated and will be removed from panda
s in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
C:\Users\Michael\AppData\Local\Temp\ipykernel_29436\3325101321.py:25: FutureW
arning: The series.append method is deprecated and will be removed from panda
s in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
C:\Users\Michael\AppData\Local\Temp\ipykernel_29436\3325101321.py:25: FutureW
arning: The series.append method is deprecated and will be removed from panda
s in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
```

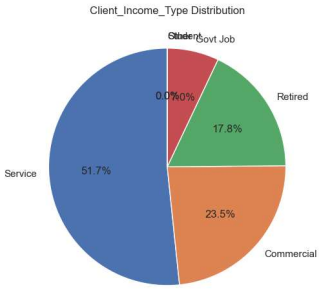

Car_Owned Distribution
yes
34.3%
65.7%
no

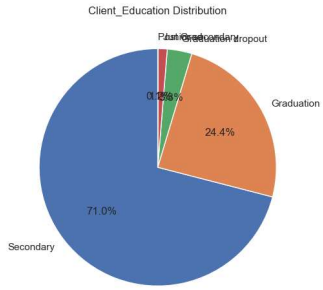

Bike_Owned Distribution
yes
33.2%
66.8%
no

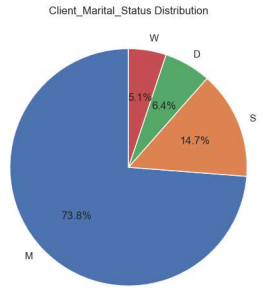

Active_Loan Distribution
no
50.1%
49.9%
yes

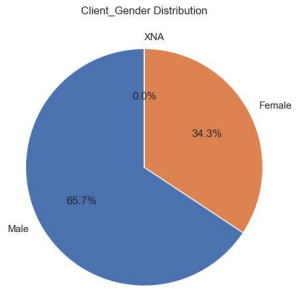

House_Own Distribution
no
30.8%
69.2%
yes

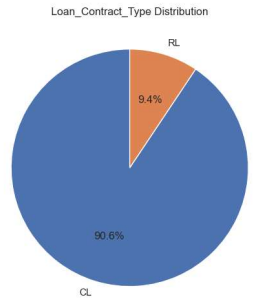

Accompany_Client Distribution
Partner
Relative
13.1%
81.1%
Alone

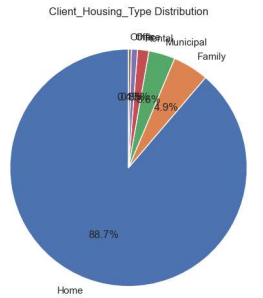

Client_Income_Type Distribution
Govt Job
Retired
17.8%
Service
51.7%
23.5%
Commercial

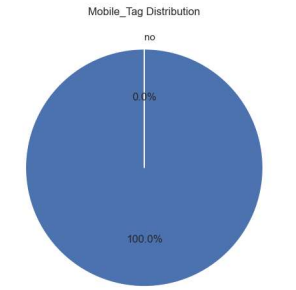

Client_Education Distribution
Graduation
24.4%
71.0%
Secondary

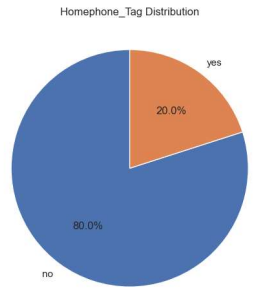

Client_Marital_Status Distribution
W
D
S
14.7%
73.8%
M

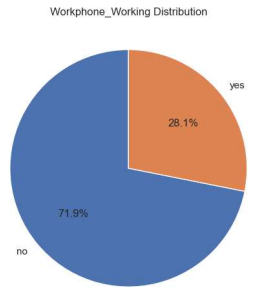

Client_Gender Distribution
XNA
0.0%
Female
34.3%
65.7%
Male

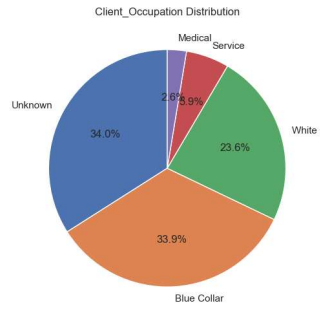

Loan_Contract_Type Distribution
RL
9.4%
90.6%
CL

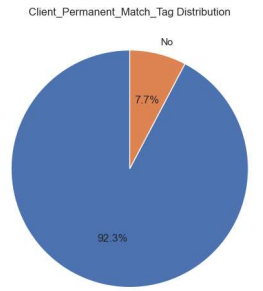

Client_Housing_Type Distribution
Municipal
Family
4.9%
88.7%
Home

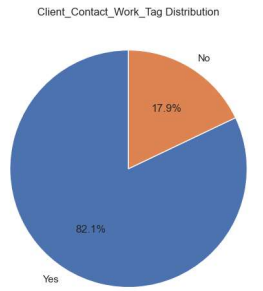

Mobile_Tag Distribution
no
0.0%
100.0%
yes

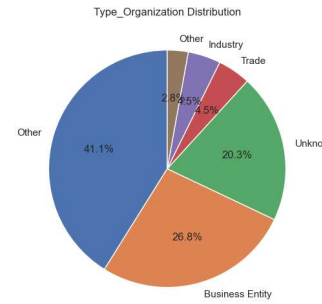

Homephone_Tag Distribution
yes
20.0%
80.0%
no


Workphone_Working Distribution
yes
28.1%
71.9%
no

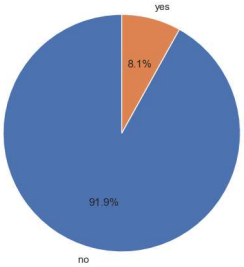

Client_Occupation Distribution
Medical
Service
Unknown
34.0%
White Collar
23.6%
33.9%
Blue Collar


Client_Permanent_Match_Tag Distribution
No
7.7%
92.3%
Yes


Client_Contact_Work_Tag Distribution
No
17.9%
82.1%
Yes


Type_Organization Distribution
Other
Industry
Trade
4.5%
Other
41.1%
Unknown
20.3%
26.8%
Business Entity

Default Distribution



Data Preprocessing Part 2

In [25]:
```
# Drop Mobile_Tag column because it only has 1 unique value
df.drop(columns = 'Mobile_Tag', inplace=True)
df.shape
```

Out[25]:
```
(121856, 37)
```

In [26]:
```
# Check the amounnt of missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[26]:
```
Score_Source_3                89.477744
Credit_Amount                 68.226431
Own_House_Age                 65.729221
Score_Source_1                56.488806
Social_Circle_Default         50.820641
Population_Region_Relative    35.477121
Loan_Annuity                  22.565980
Client_Income                 16.018908
Credit_Bureau                 15.214680
Employed_Days                  8.724232
Registration_Days              8.689765
Age_Days                       8.684020
Score_Source_2                 4.666163
Client_Income_Type             3.037191
Client_Housing_Type            3.025702
Phone_Change                   3.006828
Application_Process_Hour       3.006007
House_Own                      3.004366
Loan_Contract_Type             2.996159
Client_Education               2.991236
Child_Count                    2.985491
Active_Loan                    2.983029
Bike_Owned                     2.974002
Car_Owned                      2.938715
Client_Marital_Status          2.850085
Application_Process_Day        1.992516
Client_Gender                  1.980206
Client_Family_Members          1.977744
Cleint_City_Rating             1.976924
Accompany_Client               1.432839
dtype: float64
```

In [27]:
```
# Drop column that have missing value higher than 20%, including Client_Occupat
df.drop(columns = ['Client_Occupation', 'Score_Source_3', 'Credit_Amount', 'Ow
                   'Score_Source_1', 'Social_Circle_Default', 'Population_Regi
df.shape
```

Out[27]:
```
(121856, 29)
```

In [28]:
```
# Fill Client_Income, Employed_Days, Registration_Days, Age_Days and Credit_Bu
df['Client_Income'].fillna(df['Client_Income'].median(), inplace=True)
df['Credit_Bureau'].fillna(df['Credit_Bureau'].median(), inplace=True)
df['Employed_Days'].fillna(df['Employed_Days'].median(), inplace=True)
df['Registration_Days'].fillna(df['Registration_Days'].median(), inplace=True)
df['Age_Days'].fillna(df['Age_Days'].median(), inplace=True)
```

In [29]:
```
# Drop all of the null value for the rest of column
df.dropna(inplace=True)
df.shape
```

Out[29]:
```
(73550, 29)
```

Label Encoding for Object Datatypes

In [30]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

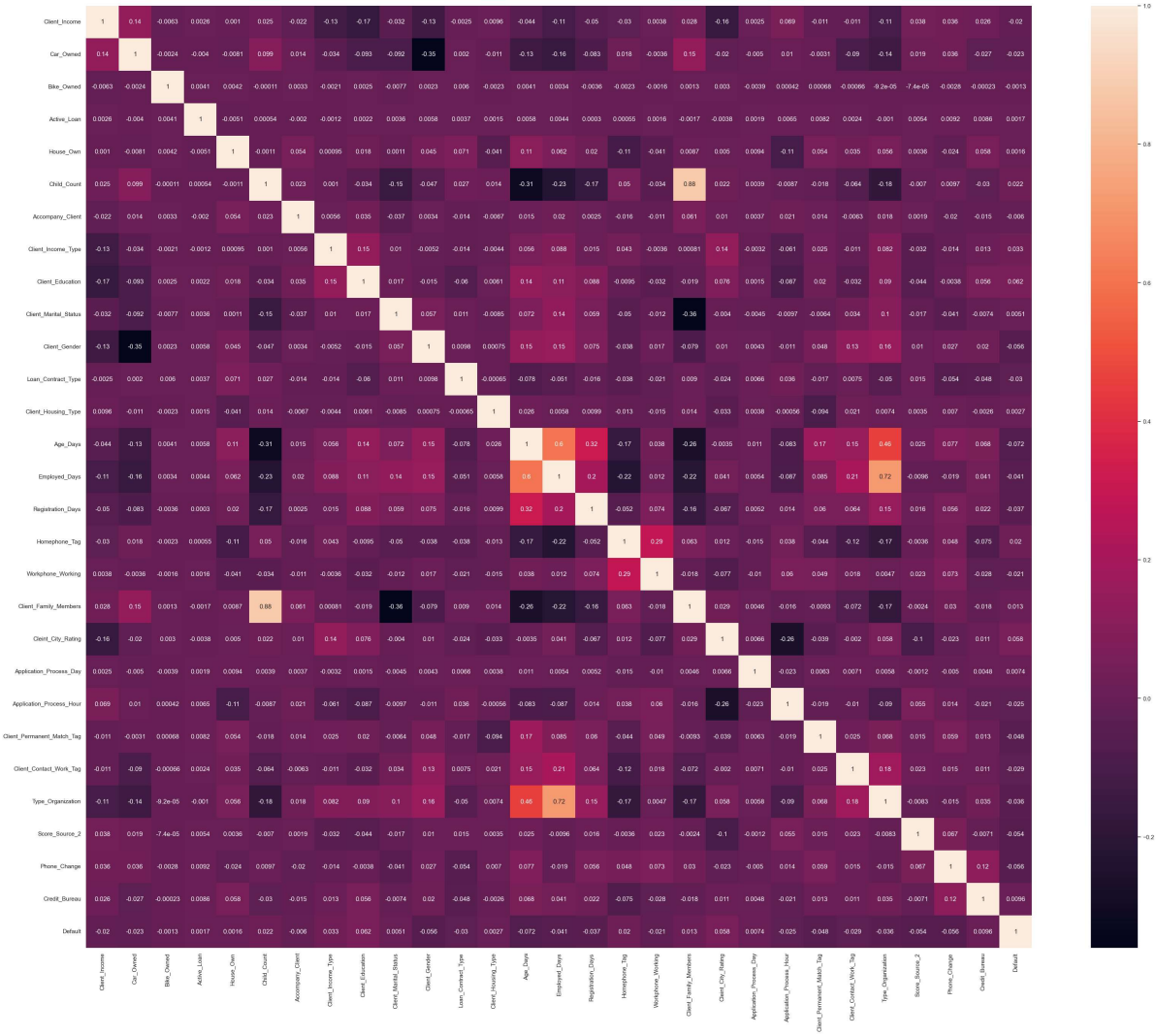
```
Car_Owned: ['no' 'yes']
Bike_Owned: ['no' 'yes']
Active_Loan: ['yes' 'no']
House_Own: ['no' 'yes']
Accompany_Client: ['Alone' 'Relative' 'Others' 'Kids' 'Partner' 'Group' '##']
Client_Income_Type: ['Commercial' 'Retired' 'Service' 'Govt Job' 'Student' 'U
nemployed'
 'Maternity leave' 'Businessman']
Client_Education: ['Secondary' 'Graduation' 'Graduation dropout' 'Junior seco
ndary'
 'Post Grad']
Client_Marital_Status: ['M' 'W' 'S' 'D']
Client_Gender: ['Male' 'Female' 'XNA']
Loan_Contract_Type: ['CL' 'RL']
Client_Housing_Type: ['Home' 'Family' 'Office' 'Municipal' 'Rental' 'Shared']
Homephone_Tag: ['yes' 'no']
Workphone_Working: ['no' 'yes']
Client_Permanent_Match_Tag: ['Yes' 'No']
Client_Contact_Work_Tag: ['Yes' 'No']
Type_Organization: ['Other' 'Unknown' 'Business Entity' 'Industry' 'Trade' 'T
ransport']
Default: ['no' 'yes']
```

In [31]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Car_Owned: [0 1]
Bike_Owned: [0 1]
Active_Loan: [1 0]
House_Own: [0 1]
Accompany_Client: [1 6 4 3 5 2 0]
Client_Income_Type: [1 4 5 2 6 7 3 0]
Client_Education: [4 0 1 2 3]
Client_Marital_Status: [1 3 2 0]
Client_Gender: [1 0 2]
Loan_Contract_Type: [0 1]
Client_Housing_Type: [1 0 3 2 4 5]
Homephone_Tag: [1 0]
Workphone_Working: [0 1]
Client_Permanent_Match_Tag: [1 0]
Client_Contact_Work_Tag: [1 0]
Type_Organization: [2 5 0 1 3 4]
Default: [0 1]
```

In [32]:
```
# Correlation Heatmap
plt.figure(figsize=(40, 32))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[32]: <AxesSubplot:>



In [33]:
```
# Drop Child_Count column because it have high correlation with Client_Family_M
df.drop(columns = 'Child_Count', inplace=True)
df.shape
```

Out[33]: (73550, 28)

In [35]:
```
df.head()
```

Out[35]:

	Client_Income	Car_Owned	Bike_Owned	Active_Loan	House_Own	Accompany_Client	Client_
0	6750.0	0	0	1	0	1	
3	15750.0	0	0	1	1	1	
5	11250.0	0	1	1	1	1	
6	15750.0	1	1	0	1	1	
7	13500.0	0	0	1	1	1	

Train Test Split

In [34]:
```
X = df.drop('Default', axis=1)
y = df['Default']
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_
```

Remove Outlier from Train Data using Z-Score

In [37]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['Client_Income', 'Employed_Days', 'Registration_Days',
                    'Client_Family_Members', 'Application_Process_Hour', 'Scor
                    'Phone_Change','Credit_Bureau']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

Decision Tree Classifier

In [57]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=9, min_samples_leaf=2
dtree.fit(X_train, y_train)
```

Out[57]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=9, min_samples_leaf
=2,
                       min_samples_split=3, random_state=0)
```

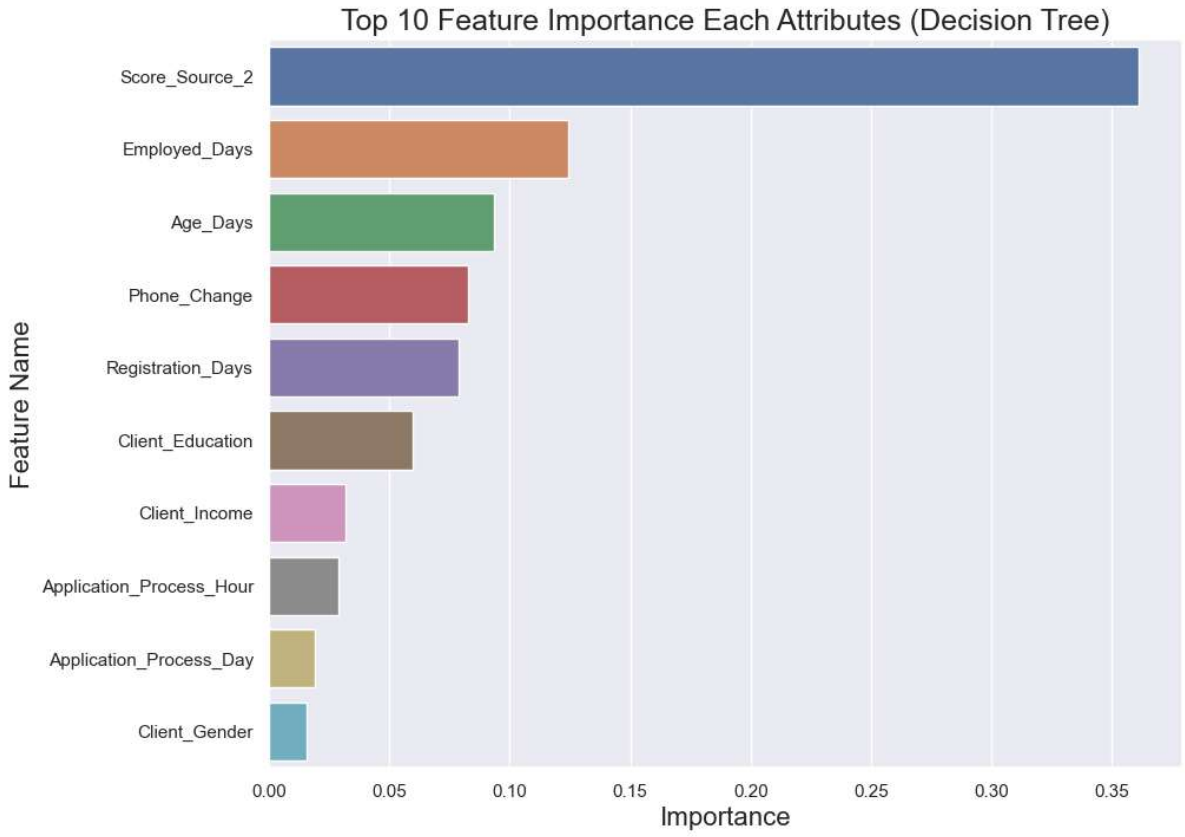
In [58]:
```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 61.3 %
```

In [59]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

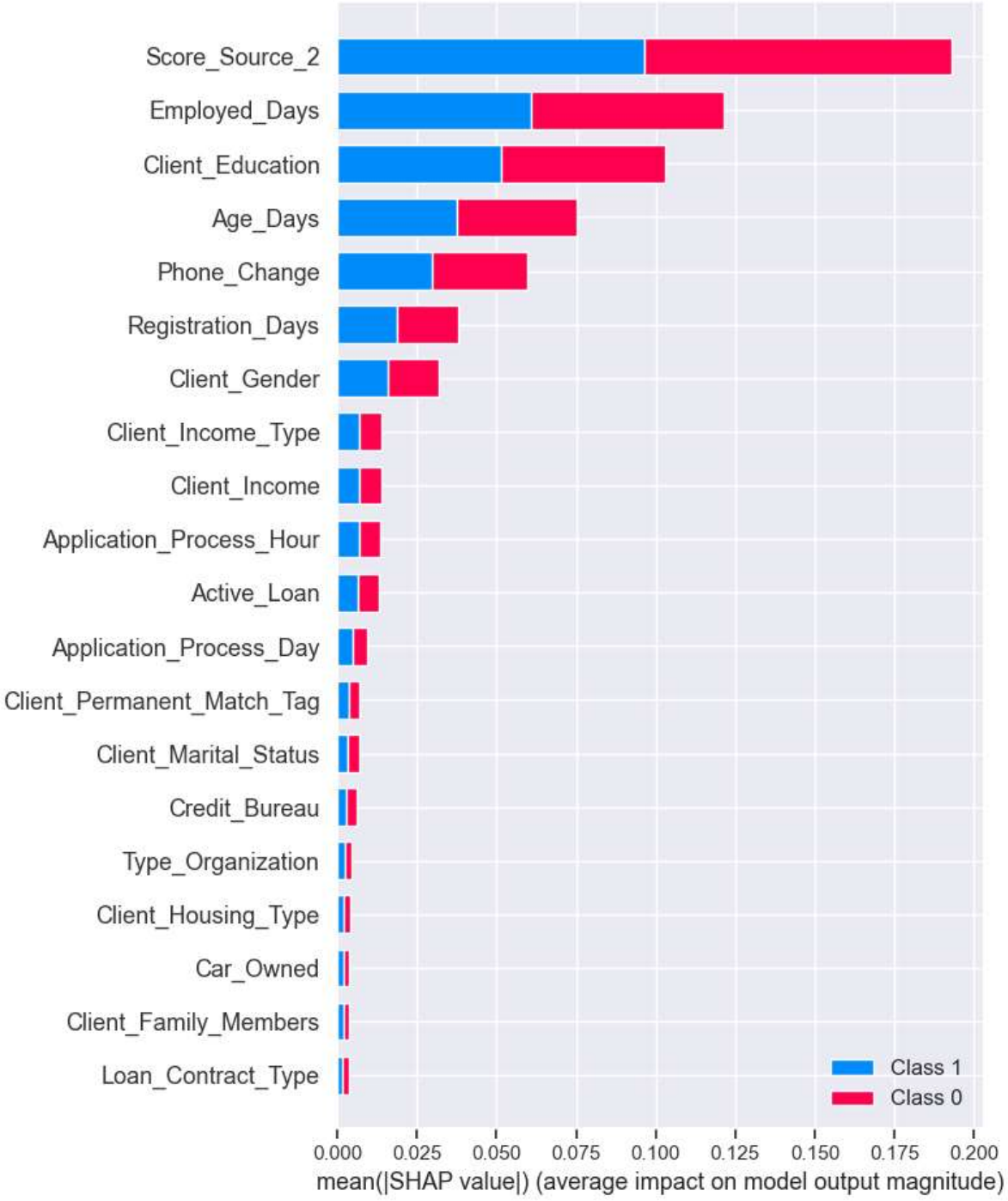
```
F-1 Score :  0.6129843643779742
Precision Score :  0.6129843643779742
Recall Score :  0.6129843643779742
Jaccard Score :  0.44194481203744546
Log Loss :  13.36733046316804
```

In [60]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Decision Tree)', fontsiz
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```

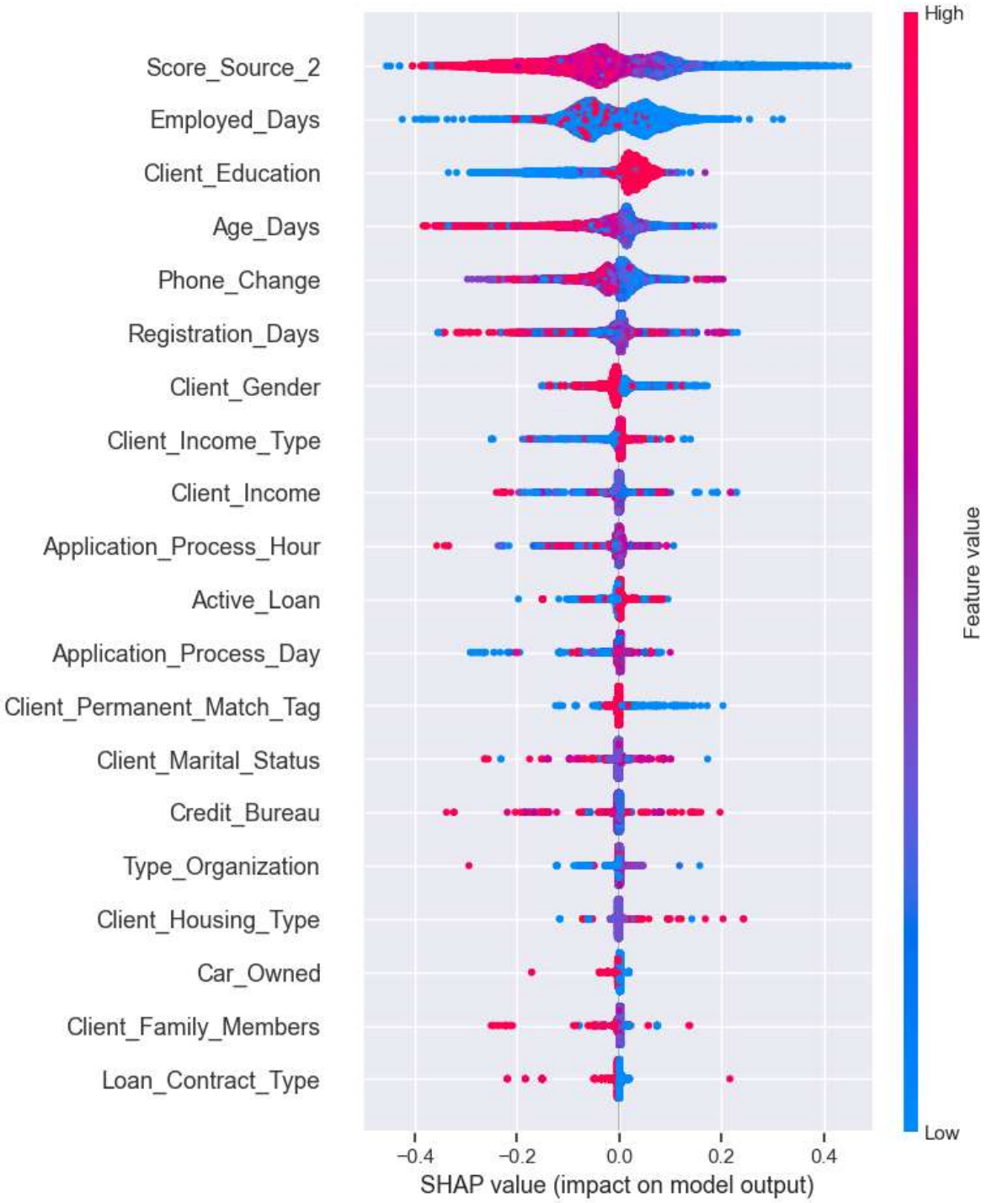


In [61]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

```
Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to f
orce console mode (e.g. in jupyter console)
```

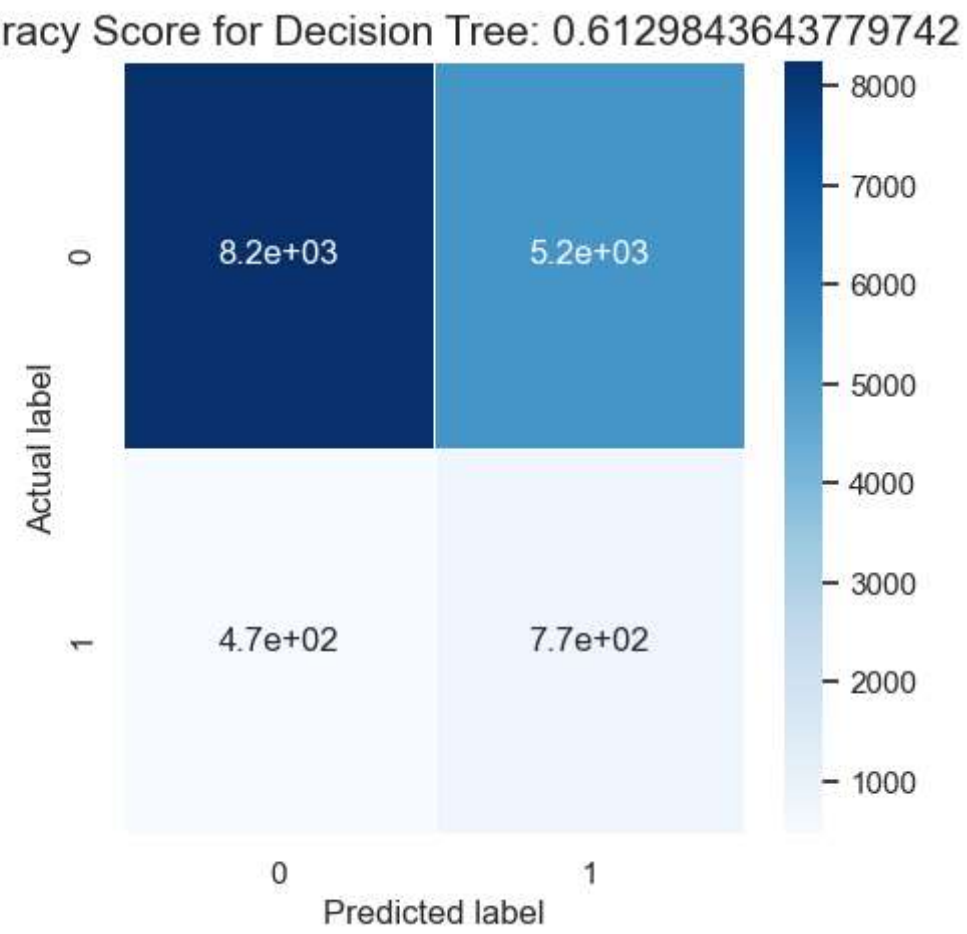


In [62]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns
```



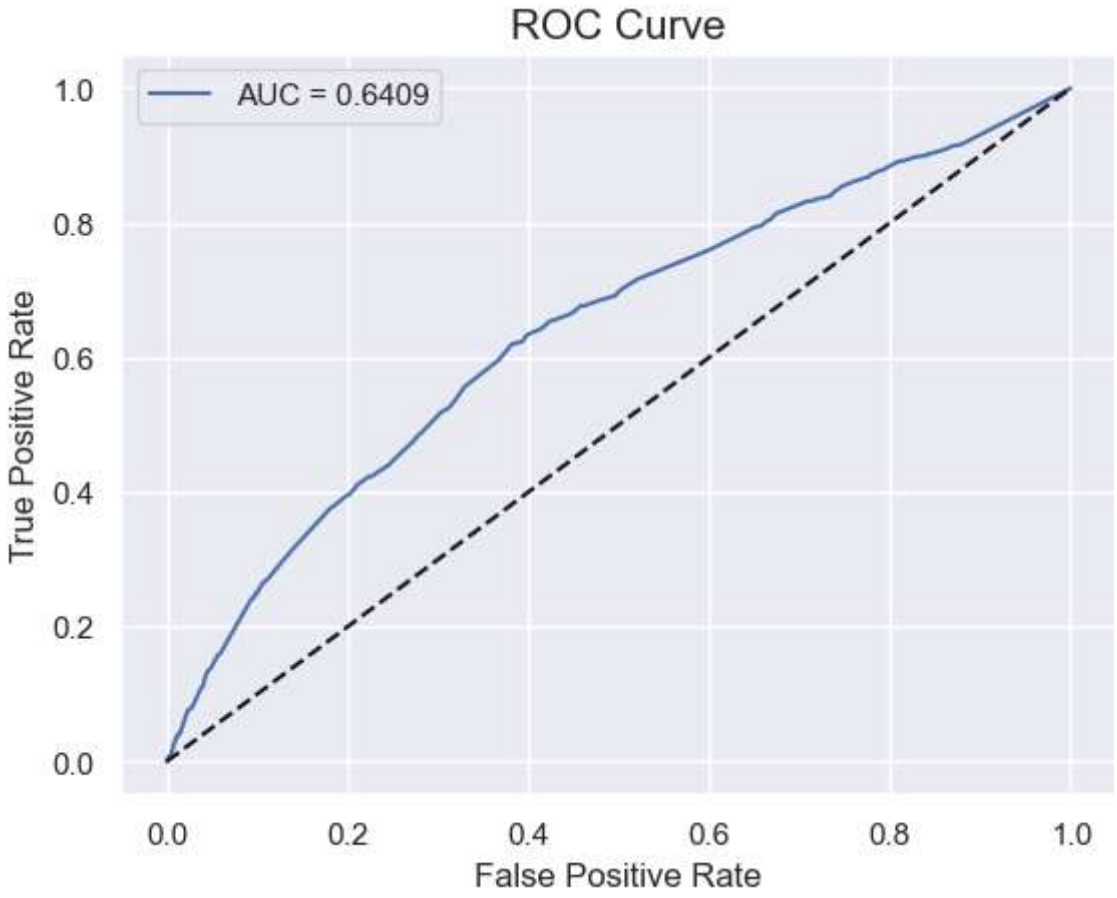
In [63]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(
plt.title(all_sample_title, size = 15)
```

Out[63]: Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.6129843643779742')



In [64]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_ac
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted[
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pr

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size = 15)
plt.legend()
```

Out[64]: <matplotlib.legend.Legend at 0x225ebaeae20>



Random Forest Classifier

In [71]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_depth=None, max_features=None
rfc.fit(X_train, y_train)
```

Out[71]:
```
RandomForestClassifier(class_weight='balanced', max_features=None,
                       random_state=0)
```

In [72]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 92.09 %
```

In [73]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.9208701563562203
Precision Score :  0.9208701563562203
Recall Score :  0.9208701563562203
Jaccard Score :  0.8533450926042585
Log Loss :  2.73304835627859
```

In [74]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsiz
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```