ml-project

October 30, 2023

##importing libraries

[105]:
```
import pandas as pd
import numpy as np
import math
import seaborn as sns
import matplotlib.pyplot as plt
import scipy
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, precision_score,␣
↪recall_score,f1_score, confusion_matrix, ConfusionMatrixDisplay,␣
↪classification_report
from sklearn.metrics import roc_auc_score, roc_curve
from imblearn.over_sampling import SMOTENC, SMOTEN
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder
from xgboost import XGBClassifier
```

##DATA PREPROCESSING

[106]:
```
df_raw= pd.read_csv("/content/Credit_card.csv")
```

[107]:
```
df_credit_raw = pd.read_csv("/content/Credit_card_label.csv", encoding =␣
↪'utf-8')
```

[108]:
```
df=df_raw.copy()
```

[109]:
```
df_credit=df_credit_raw.copy()
```

[110]:
```
df.head()
```

[110]:
```
    Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
0  5008827      M         Y             Y         0       180000.0
1  5009744      F         Y             N         0       315000.0
2  5009746      F         Y             N         0       315000.0
3  5009749      F         Y             N         0            NaN
4  5009752      F         Y             N         0       315000.0
```

```
           Type_Income         EDUCATION Marital_status       Housing_type  \
0            Pensioner  Higher education        Married  House / apartment
1  Commercial associate  Higher education        Married  House / apartment
2  Commercial associate  Higher education        Married  House / apartment
3  Commercial associate  Higher education        Married  House / apartment
4  Commercial associate  Higher education        Married  House / apartment

   Birthday_count  Employed_days  Mobile_phone  Work_Phone  Phone  EMAIL_ID  \
0        -18772.0         365243             1           0      0         0
1        -13557.0           -586             1           1      1         0
2             NaN           -586             1           1      1         0
3        -13557.0           -586             1           1      1         0
4        -13557.0           -586             1           1      1         0

  Type_Occupation  Family_Members
0             NaN               2
1             NaN               2
2             NaN               2
3             NaN               2
4             NaN               2
```

[111]:
```
df_credit.head()
```

[111]:
```
    Ind_ID  label
0  5008827      1
1  5009744      1
2  5009746      1
3  5009749      1
4  5009752      1
```

[112]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Ind_ID           1548 non-null   int64
 1   GENDER           1541 non-null   object
 2   Car_Owner        1548 non-null   object
 3   Propert_Owner    1548 non-null   object
 4   CHILDREN         1548 non-null   int64
 5   Annual_income    1525 non-null   float64
 6   Type_Income      1548 non-null   object
 7   EDUCATION        1548 non-null   object
 8   Marital_status   1548 non-null   object
```

```
 9   Housing_type     1548 non-null   object
 10  Birthday_count   1526 non-null   float64
 11  Employed_days    1548 non-null   int64
 12  Mobile_phone     1548 non-null   int64
 13  Work_Phone       1548 non-null   int64
 14  Phone            1548 non-null   int64
 15  EMAIL_ID         1548 non-null   int64
 16  Type_Occupation  1060 non-null   object
 17  Family_Members   1548 non-null   int64
dtypes: float64(2), int64(8), object(8)
memory usage: 217.8+ KB
```

[113]:
```
print('For the first table, number of unique ID',df['Ind_ID'].nunique())
print('For the second table, number of unique ID',df_credit['Ind_ID'].nunique())
print('Number of unique customer ID that appearing in both tables:
 ↪',df[df['Ind_ID'].isin(df_credit['Ind_ID'])]['Ind_ID'].nunique())
```

```
For the first table, number of unique ID 1548
For the second table, number of unique ID 1548
Number of unique customer ID that appearing in both tables: 1548
```

[114]:
```
df.shape
```

[114]:
```
(1548, 18)
```

[115]:
```
df_credit.shape
```

[115]:
```
(1548, 2)
```

[116]:
```
df.isnull().sum()
```

[116]:
```
Ind_ID              0
GENDER              7
Car_Owner           0
Propert_Owner       0
CHILDREN            0
Annual_income      23
Type_Income         0
EDUCATION           0
Marital_status      0
Housing_type        0
Birthday_count     22
Employed_days       0
Mobile_phone        0
Work_Phone          0
Phone               0
EMAIL_ID            0
Type_Occupation   488
```

```
Family_Members       0
dtype: int64
```

[117]:
```
df_credit.isnull().sum()
```

[117]:
```
Ind_ID    0
label     0
dtype: int64
```

[118]:
```
df.nunique()
```

[118]:
```
Ind_ID              1548
GENDER                 2
Car_Owner              2
Propert_Owner          2
CHILDREN               6
Annual_income        115
Type_Income            4
EDUCATION              5
Marital_status         5
Housing_type           6
Birthday_count      1270
Employed_days        956
Mobile_phone           1
Work_Phone             2
Phone                  2
EMAIL_ID               2
Type_Occupation       18
Family_Members         7
dtype: int64
```

[119]:
```
df.describe(include='all')
```

[119]:
```
              Ind_ID GENDER Car_Owner Propert_Owner     CHILDREN  \
count   1.548000e+03   1541      1548          1548  1548.000000
unique           NaN      2         2             2          NaN
top              NaN      F         N             Y          NaN
freq             NaN    973       924          1010          NaN
mean    5.078920e+06    NaN       NaN           NaN     0.412791
std     4.171759e+04    NaN       NaN           NaN     0.776691
min     5.008827e+06    NaN       NaN           NaN     0.000000
25%     5.045070e+06    NaN       NaN           NaN     0.000000
50%     5.078842e+06    NaN       NaN           NaN     0.000000
75%     5.115673e+06    NaN       NaN           NaN     1.000000
max     5.150412e+06    NaN       NaN           NaN    14.000000

        Annual_income Type_Income                     EDUCATION  \
```

count	1.525000e+03	1548	1548
unique	NaN	4	5
top	NaN	Working	Secondary / secondary special
freq	NaN	798	1031
mean	1.913993e+05	NaN	NaN
std	1.132530e+05	NaN	NaN
min	3.375000e+04	NaN	NaN
25%	1.215000e+05	NaN	NaN
50%	1.665000e+05	NaN	NaN
75%	2.250000e+05	NaN	NaN
max	1.575000e+06	NaN	NaN

	Marital_status	Housing_type	Birthday_count	Employed_days \
count	1548	1548	1526.000000	1548.000000
unique	5	6	NaN	NaN
top	Married	House / apartment	NaN	NaN
freq	1049	1380	NaN	NaN
mean	NaN	NaN	-16040.342071	59364.689922
std	NaN	NaN	4229.503202	137808.062701
min	NaN	NaN	-24946.000000	-14887.000000
25%	NaN	NaN	-19553.000000	-3174.500000
50%	NaN	NaN	-15661.500000	-1565.000000
75%	NaN	NaN	-12417.000000	-431.750000
max	NaN	NaN	-7705.000000	365243.000000

	Mobile_phone	Work_Phone	Phone	EMAIL_ID	Type_Occupation \
count	1548.0	1548.000000	1548.000000	1548.000000	1060
unique	NaN	NaN	NaN	NaN	18
top	NaN	NaN	NaN	NaN	Laborers
freq	NaN	NaN	NaN	NaN	268
mean	1.0	0.208010	0.309432	0.092377	NaN
std	0.0	0.406015	0.462409	0.289651	NaN
min	1.0	0.000000	0.000000	0.000000	NaN
25%	1.0	0.000000	0.000000	0.000000	NaN
50%	1.0	0.000000	0.000000	0.000000	NaN
75%	1.0	0.000000	1.000000	0.000000	NaN
max	1.0	1.000000	1.000000	1.000000	NaN

	Family_Members
count	1548.000000
unique	NaN
top	NaN
freq	NaN
mean	2.161499
std	0.947772
min	1.000000
25%	2.000000

```
50%            2.000000
75%            3.000000
max           15.000000
```

[120]:
```
join_data = pd.merge(df,df_credit)
join_data.head()
```

[120]:
```
    Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
0  5008827      M         Y             Y         0       180000.0
1  5009744      F         Y             N         0       315000.0
2  5009746      F         Y             N         0       315000.0
3  5009749      F         Y             N         0            NaN
4  5009752      F         Y             N         0       315000.0

            Type_Income         EDUCATION Marital_status       Housing_type  \
0             Pensioner  Higher education        Married  House / apartment
1  Commercial associate  Higher education        Married  House / apartment
2  Commercial associate  Higher education        Married  House / apartment
3  Commercial associate  Higher education        Married  House / apartment
4  Commercial associate  Higher education        Married  House / apartment

   Birthday_count  Employed_days  Mobile_phone  Work_Phone  Phone  EMAIL_ID  \
0        -18772.0         365243             1           0      0         0
1        -13557.0           -586             1           1      1         0
2             NaN           -586             1           1      1         0
3        -13557.0           -586             1           1      1         0
4        -13557.0           -586             1           1      1         0

  Type_Occupation  Family_Members  label
0             NaN               2      1
1             NaN               2      1
2             NaN               2      1
3             NaN               2      1
4             NaN               2      1
```

[121]:
```
join_data.isnull().sum()
```

[121]:
```
Ind_ID              0
GENDER              7
Car_Owner           0
Propert_Owner       0
CHILDREN            0
Annual_income      23
Type_Income         0
EDUCATION           0
Marital_status      0
Housing_type        0
```

```
Birthday_count      22
Employed_days        0
Mobile_phone         0
Work_Phone           0
Phone                0
EMAIL_ID             0
Type_Occupation    488
Family_Members       0
label                0
dtype: int64
```

###Set DAYS_BIRTH, DAYS_EMPLOTED to a more appropriate format

[122]:
```
join_data[join_data['Birthday_count']==0]
```

[122]:
```
Empty DataFrame
Columns: [Ind_ID, GENDER, Car_Owner, Propert_Owner, CHILDREN, Annual_income,
Type_Income, EDUCATION, Marital_status, Housing_type, Birthday_count,
Employed_days, Mobile_phone, Work_Phone, Phone, EMAIL_ID, Type_Occupation,
Family_Members, label]
Index: []
```

[123]:
```
join_data['Birthday_count'] = join_data['Birthday_count'].fillna(0)
```

[124]:
```
join_data['Birthday_count'] = round(join_data['Birthday_count']/-365,0)
join_data.rename(columns={'Birthday_count':'AGE'}, inplace=True)
```

[125]:
```
join_data[join_data['Employed_days']>0]['Employed_days'].unique()
```

[125]:
```
array([365243])
```

[126]:
```
# As mentioned in document, if 'Employed_days' is positive no, it means person␣
↪currently unemployed, hence replacing it with 0
join_data['Employed_days'].replace(365243, 0, inplace=True)
```

[127]:
```
join_data['Employed_days'] = abs(round(join_data['Employed_days']/-365,0))
join_data.rename(columns={'Employed_days':'YEAR_EMPLOYED'}, inplace=True)
```

[128]:
```
join_data= join_data.reset_index(drop=True)
```

[129]:
```
# Calculate the mean value of the column, rounded to the nearest integer
mean_value = round(join_data[join_data['AGE'] != 0]['AGE'].mean())

# Replace zeros with the mean value
join_data['AGE'] = join_data['AGE'].replace(0, mean_value)
```

[130]:
```
# Example using scikit-learn's LinearRegression
from sklearn.linear_model import LinearRegression

# Split your data into two sets: one with missing 'Annual Income' and one␣
↪without
df_missing = join_data[join_data['Annual_income'].isnull()]
df_not_missing = join_data[~join_data['Annual_income'].isnull()]

# Fit a regression model
model = LinearRegression()
model.fit(df_not_missing[['AGE','YEAR_EMPLOYED']],␣
↪df_not_missing['Annual_income'])

# Predict missing values
predicted_incomes = model.predict(df_missing[['AGE','YEAR_EMPLOYED']])

# Round predicted incomes to integers
predicted_incomes = predicted_incomes.round().astype(int)

# Fill missing values with predicted values
join_data.loc[join_data['Annual_income'].isnull(), 'Annual_income'] =␣
↪predicted_incomes
```

[131]:
```
join_data['GENDER'].fillna(join_data['GENDER'].mode()[0],inplace=True)
```

[132]:
```
join_data.loc[join_data["Type_Income"]=="Pensioner","Type_Occupation"] =␣
↪"Pensioner"
```

[133]:
```
join_data.isna().sum()
```

[133]:
```
Ind_ID             0
GENDER             0
Car_Owner          0
Propert_Owner      0
CHILDREN           0
Annual_income      0
Type_Income        0
EDUCATION          0
Marital_status     0
Housing_type       0
AGE                0
YEAR_EMPLOYED      0
Mobile_phone       0
Work_Phone         0
Phone              0
EMAIL_ID           0
Type_Occupation  224
```

```
Family_Members       0
label                0
dtype: int64
```

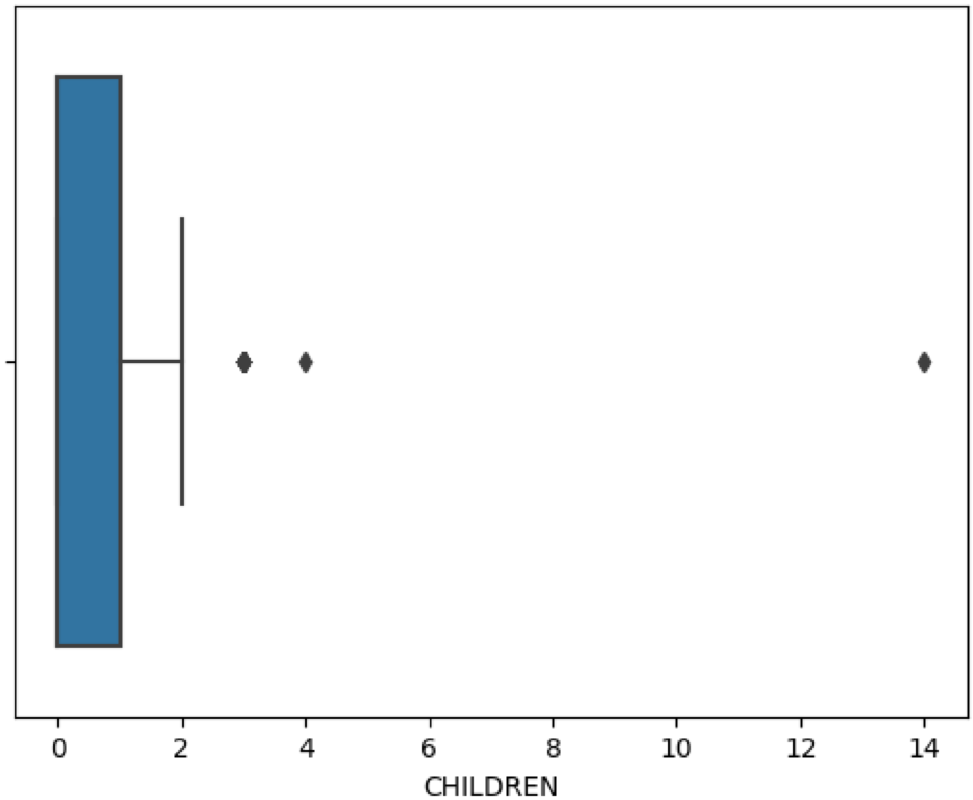
[134]:
```
join_data['Type_Occupation'].fillna('Unknown', inplace=True)
```

[135]:
```
join_data.isna().sum()
```

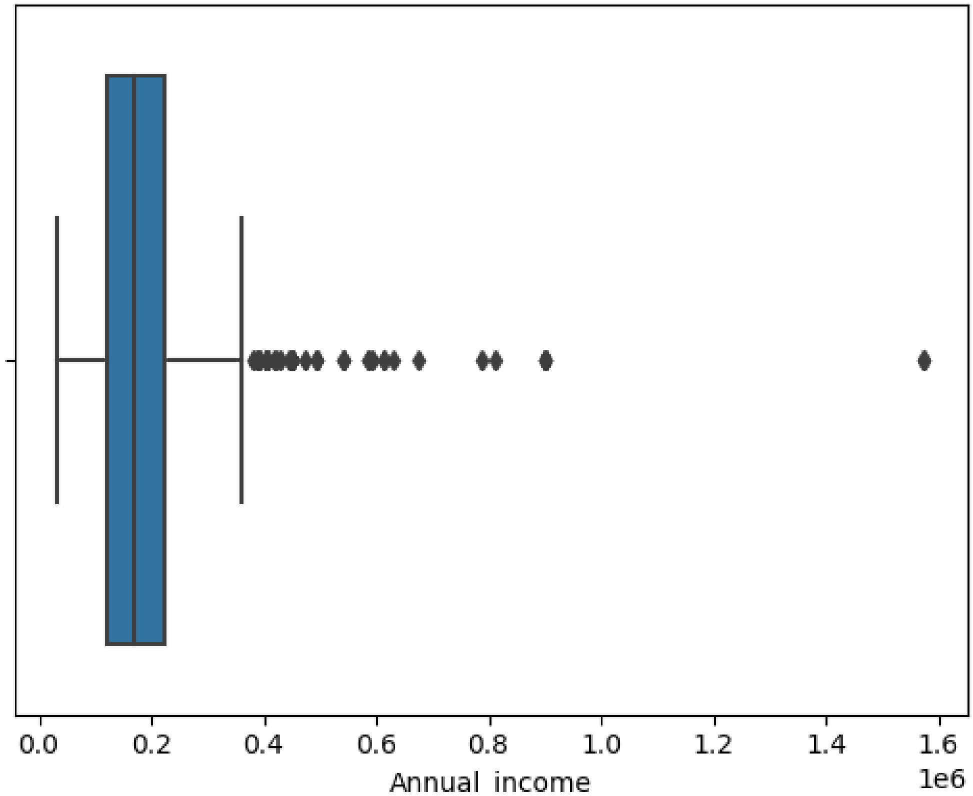
[135]:
```
Ind_ID             0
GENDER             0
Car_Owner          0
Propert_Owner      0
CHILDREN           0
Annual_income      0
Type_Income        0
EDUCATION          0
Marital_status     0
Housing_type       0
AGE                0
YEAR_EMPLOYED      0
Mobile_phone       0
Work_Phone         0
Phone              0
EMAIL_ID           0
Type_Occupation    0
Family_Members     0
label              0
dtype: int64
```

###Visualization

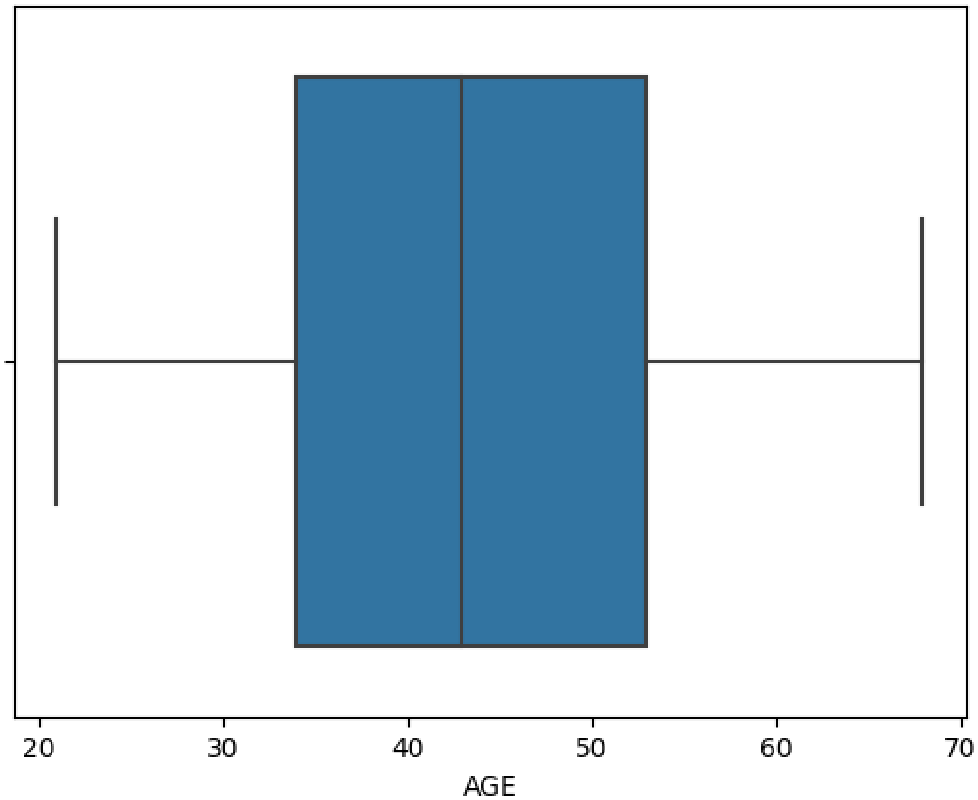
[136]:
```
#create plot to detect outliers
sns.boxplot(x=join_data['CHILDREN'])
plt.xlabel('CHILDREN')
plt.show()
```



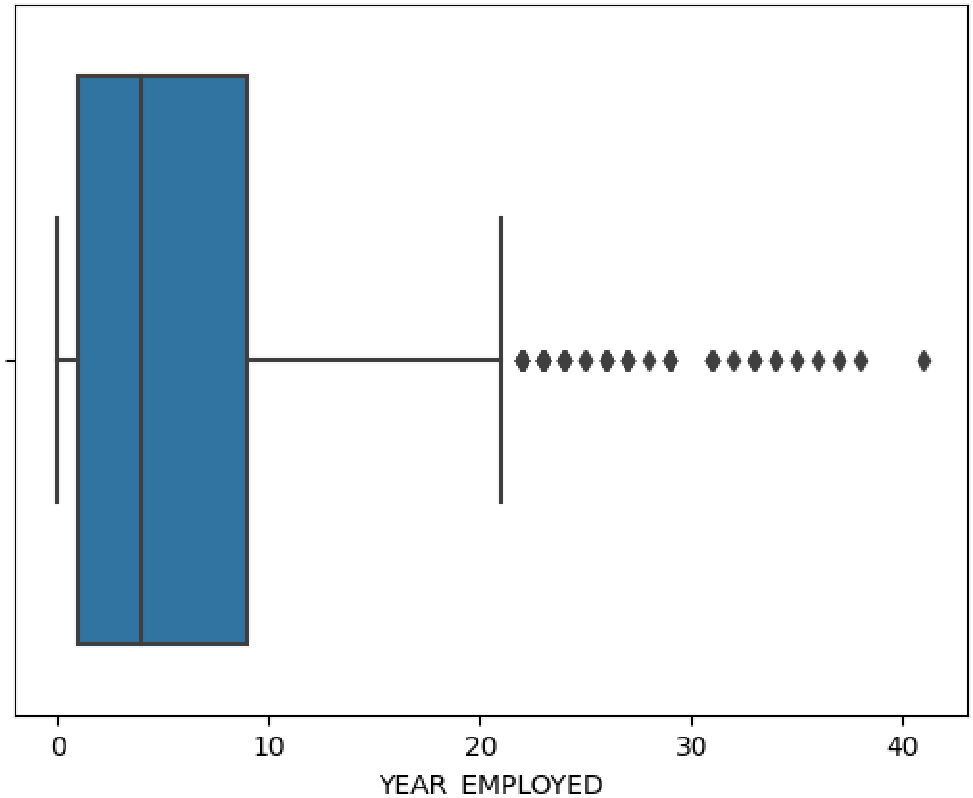
[137]:
```
#create plot to detect outliers
sns.boxplot(x=join_data['Annual_income'])
plt.xlabel('Annual_income')
plt.show()
```



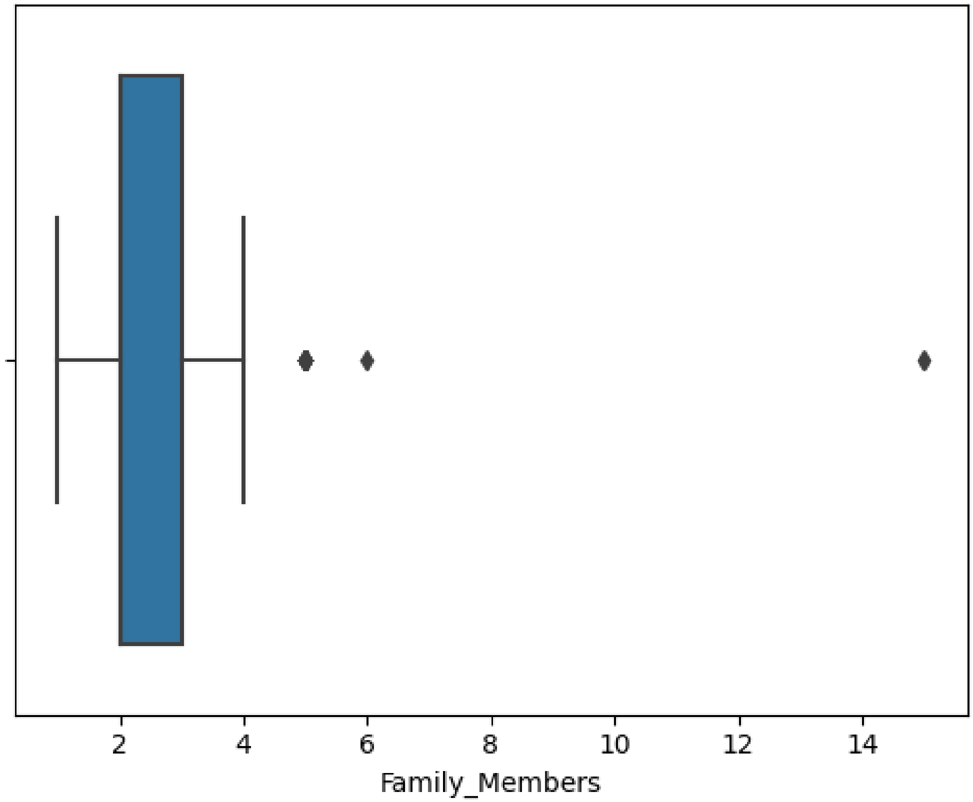
[138]:
```
#create plot to detect outliers
sns.boxplot(x=join_data['AGE'])
plt.xlabel('AGE')
plt.show()
```



[139]:
```
#create plot to detect outliers
sns.boxplot(x=join_data['YEAR_EMPLOYED'])
plt.xlabel('YEAR_EMPLOYED')
plt.show()
```



[140]:
```
#create plot to detect outliers
sns.boxplot(x=join_data['Family_Members'])
plt.xlabel('Family_Members')
plt.show()
```



###Treating Outliers

[141]:
```
import numpy as np

def detect_outliers_iqr(data):
    data = sorted(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    IQR = q3 - q1
    lower_bound = q1 - (1.5 * IQR)
    upper_bound = q3 + (1.5 * IQR)
    outliers = []

    for i in data:
        if i < lower_bound or i > upper_bound:
            outliers.append(i)

    return outliers

# 'join_data' is a DataFrame with a 'CHILDREN' column
# You can extract the 'CHILDREN' column and pass it to the function
```

```python
children_column = join_data['CHILDREN']
outliers = detect_outliers_iqr(children_column)

print("Outliers from IQR method in the 'CHILDREN' column:")
print(outliers)
```

```
Outliers from IQR method in the 'CHILDREN' column:
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 14]
```

###Median Imputation

[142]:
```python
# Calculate the median of the outliers
median_of_outliers = np.median(outliers)

# Replace outliers in the 'CHILDREN' column with the median
join_data['CHILDREN'] = join_data['CHILDREN'].apply(lambda x:␣
↪median_of_outliers if x in outliers else x)
```

[143]:
```python
import numpy as np

def detect_outliers_iqr(data):
    data = sorted(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    IQR = q3 - q1
    lower_bound = q1 - (1.5 * IQR)
    upper_bound = q3 + (1.5 * IQR)
    outliers = []

    for i in data:
        if i < lower_bound or i > upper_bound:
            outliers.append(i)

    return outliers

Annual_income_column = join_data['Annual_income']

# Detect outliers
outliers = detect_outliers_iqr(Annual_income_column)

# Calculate the median of the outliers
median_of_outliers = np.median(outliers)

# Replace outliers in the 'Annual_income' column with the median
join_data['Annual_income'] = join_data['Annual_income'].apply(lambda x:␣
↪median_of_outliers if x in outliers else x)
```

[144]:
```python
import numpy as np

def detect_outliers_iqr(data):
    data = sorted(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    IQR = q3 - q1
    lower_bound = q1 - (1.5 * IQR)
    upper_bound = q3 + (1.5 * IQR)
    outliers = []

    for i in data:
        if i < lower_bound or i > upper_bound:
            outliers.append(i)

    return outliers

YEAR_EMPLOYED_column = join_data['YEAR_EMPLOYED']

# Detect outliers
outliers = detect_outliers_iqr(YEAR_EMPLOYED_column)

# Calculate the median of the outliers
median_of_outliers = np.median(outliers)

# Replace outliers in the 'YEAR_EMPLOYED' column with the median
join_data['YEAR_EMPLOYED'] = join_data['YEAR_EMPLOYED'].apply(lambda x:
 ↪median_of_outliers if x in outliers else x)
```

[145]:
```python
import numpy as np

def detect_outliers_iqr(data):
    data = sorted(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    IQR = q3 - q1
    lower_bound = q1 - (1.5 * IQR)
    upper_bound = q3 + (1.5 * IQR)
    outliers = []

    for i in data:
        if i < lower_bound or i > upper_bound:
            outliers.append(i)

    return outliers

Family_Members_column = join_data['Family_Members']
```

```
# Detect outliers
outliers = detect_outliers_iqr(Family_Members_column)

# Calculate the median of the outliers
median_of_outliers = np.median(outliers)

# Replace outliers in the 'Family_Members' column with the median
join_data['Family_Members'] = join_data['Family_Members'].apply(lambda x:␣
↪median_of_outliers if x in outliers else x)
```

###Feature Selection

[146]:
```
join_data["Mobile_phone"].value_counts()
```

[146]:
```
1    1548
Name: Mobile_phone, dtype: int64
```

[147]:
```
# As all the values in column are 1, hence dropping column
join_data = join_data.drop('Mobile_phone',axis=1)
```

[148]:
```
join_data['Work_Phone'].value_counts()
```

[148]:
```
0    1226
1     322
Name: Work_Phone, dtype: int64
```

[149]:
```
# This column only contains 0 & 1 values for Mobile no submitted, hence␣
↪dropping column
join_data.drop('Work_Phone', axis=1, inplace=True)
```

[150]:
```
join_data['Phone'].value_counts()
```

[150]:
```
0    1069
1     479
Name: Phone, dtype: int64
```

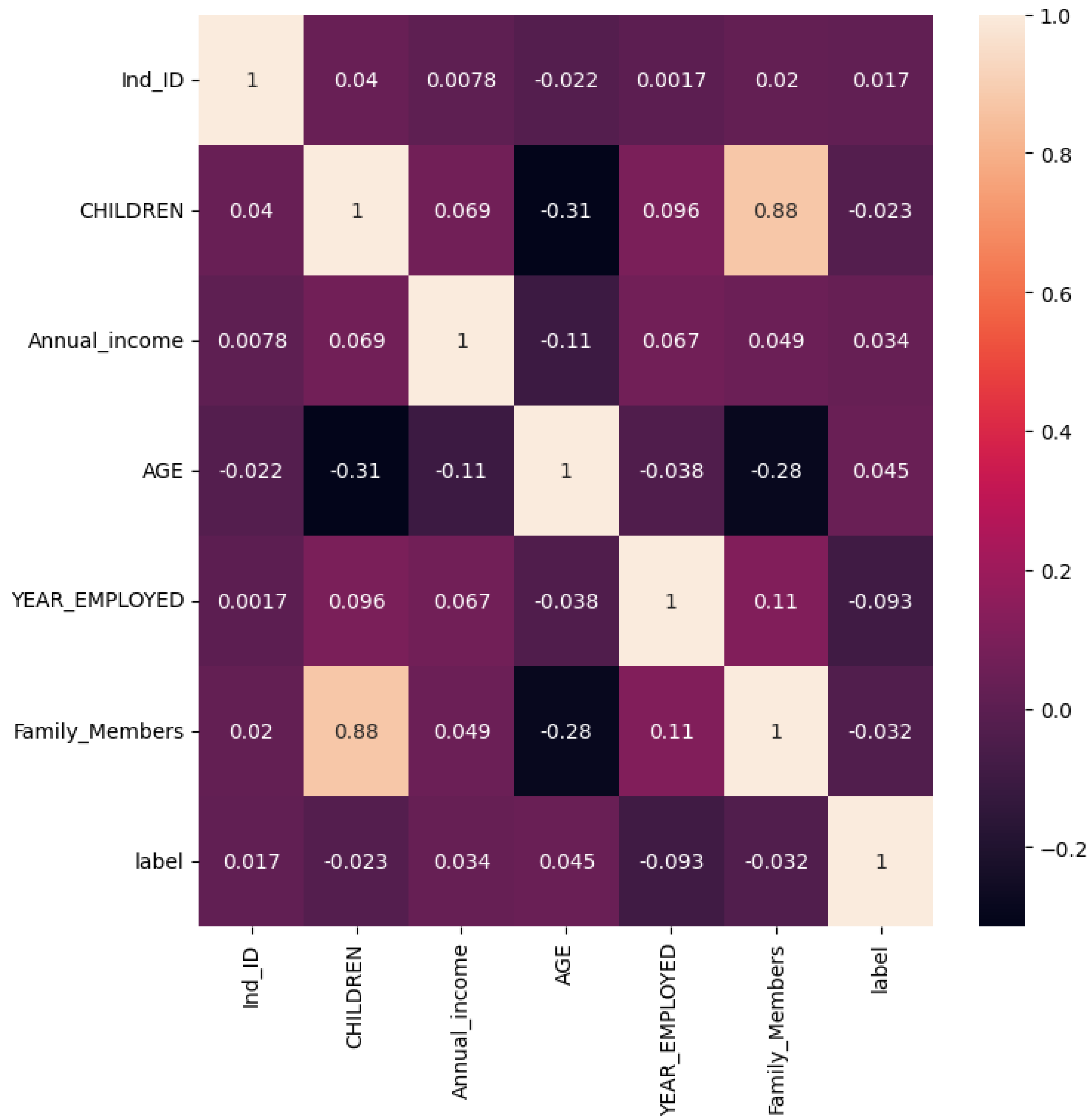
[151]:
```
# This column only contains 0 & 1 values for Mobile no submitted, hence␣
↪dropping column
join_data.drop('Phone', axis=1, inplace=True)
```

[152]:
```
join_data['EMAIL_ID'].value_counts()
```

[152]:
```
0    1405
1     143
Name: EMAIL_ID, dtype: int64
```

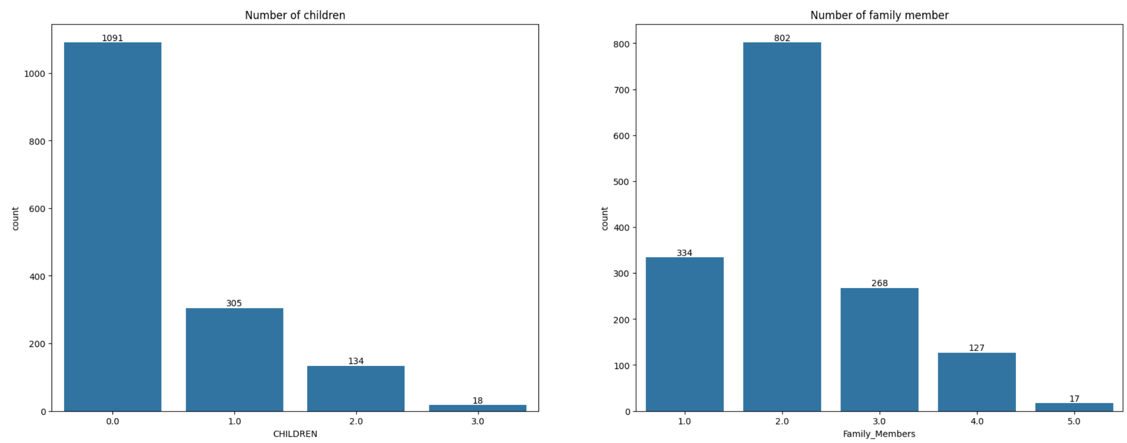
[153]:
```
# This column only contains 0 & 1 values for Email submitted, hence dropping
↪column
join_data.drop('EMAIL_ID', axis=1, inplace=True)
```

[154]:
```
# This graph shows that, there is no column (Feature) which is highly
↪co-related with 'label'
plt.figure(figsize = (8,8))
sns.heatmap(join_data.corr(numeric_only=True), annot=True)

plt.show()
```



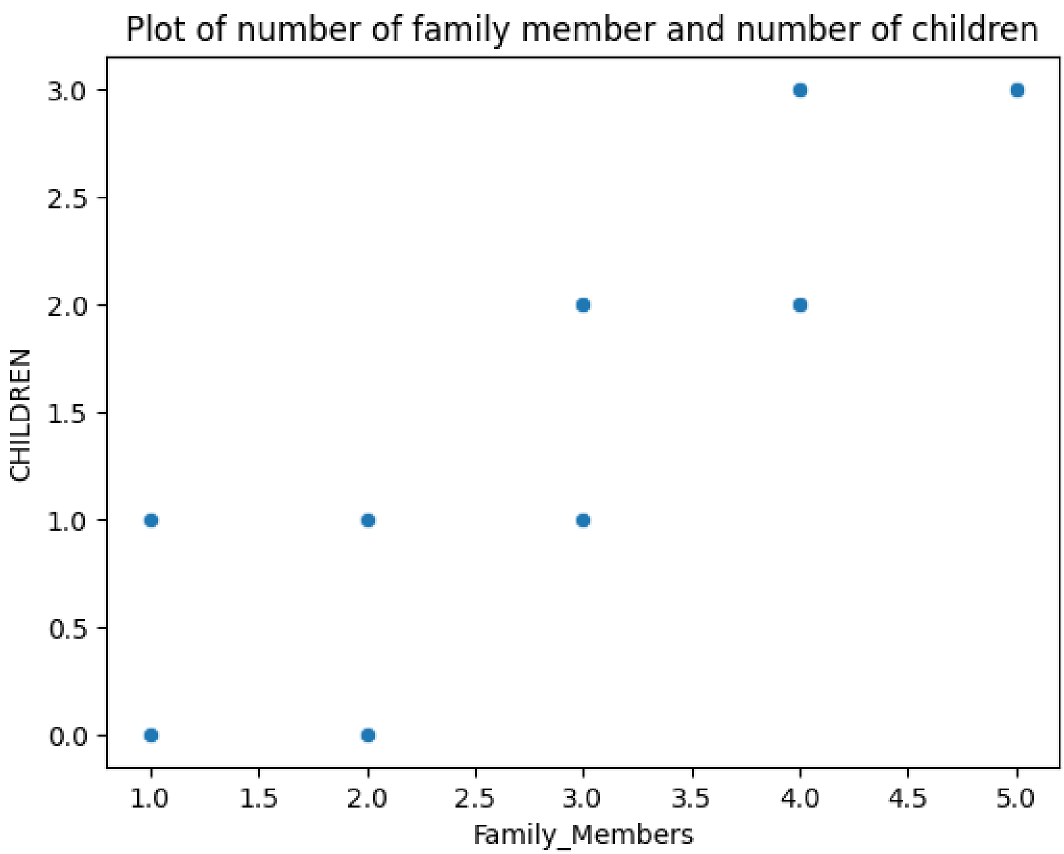
CHILDREN(Number of children) and Family_Members (Number of family member)

```
[155]: fig, ax = plt.subplots(1, 2, figsize = (22,8))
cplot = sns.countplot(data=join_data, x="CHILDREN", ax=ax[0],color='tab:Blue')
for container in cplot.containers:
    cplot.bar_label(container)
ax[0].set_title('Number of children')

cplot = sns.countplot(data=join_data, x="Family_Members", ax=ax[1],color='tab:
 ↪Blue')
for container in cplot.containers:
    cplot.bar_label(container)
ax[1].set_title('Number of family member')
plt.show()
```



There is clearly outliers on both number of children and family member. The distribution of number of family greater than 1 is exactly the same as the distribution of number of children. This shows that these two features are highly correlated.

```
[156]: sns.scatterplot(data=join_data, x="Family_Members",y="CHILDREN")
plt.title('Plot of number of family member and number of children')
plt.show()
```



[157]:
```
join_data[["CHILDREN","Family_Members"]].corr()
```

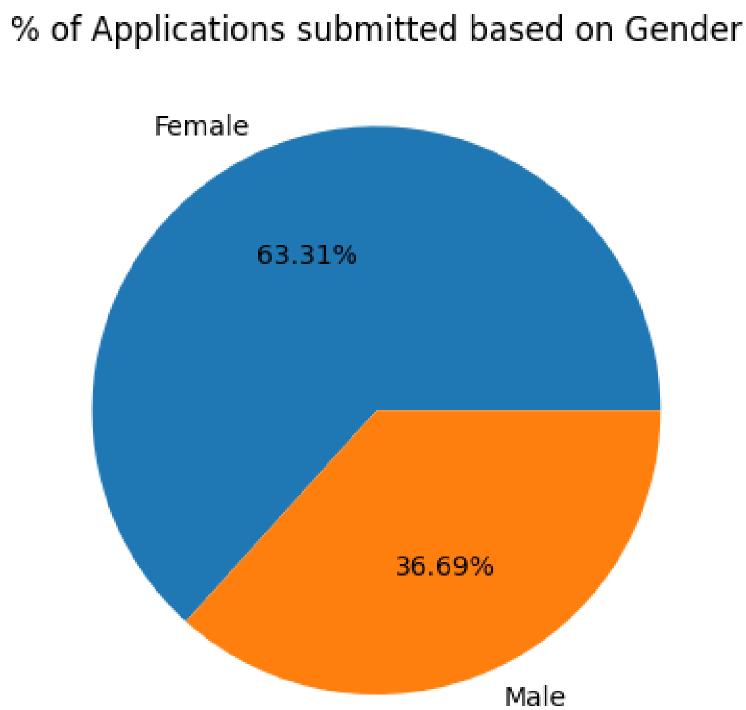
[157]:

	CHILDREN	Family_Members
CHILDREN	1.000000	0.876728
Family_Members	0.876728	1.000000

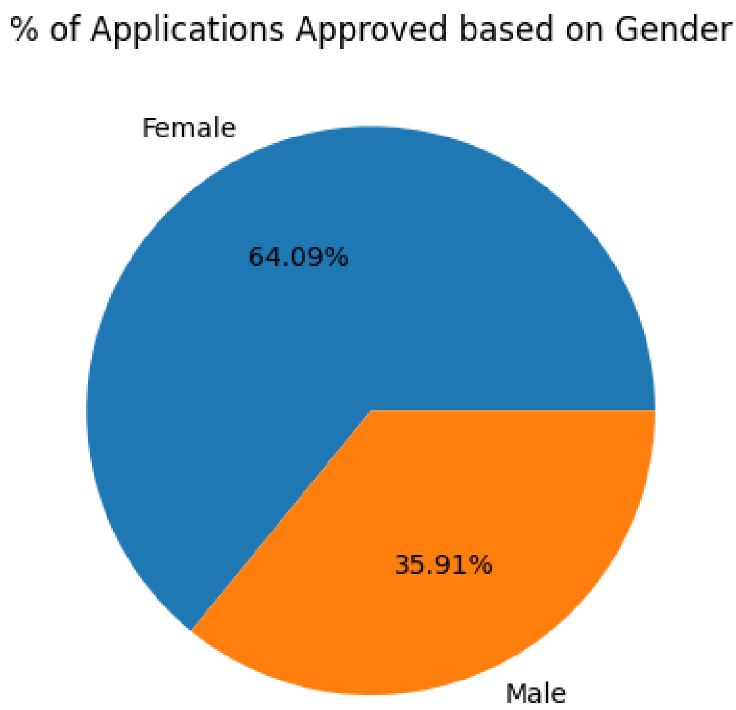
The plot of number of family member and number of children and correlation table confirm the correlation. As the number of family member cover the number of children, we chose to drop the number of children feature.

[158]:
```
join_data = join_data.drop("CHILDREN",axis=1)
```

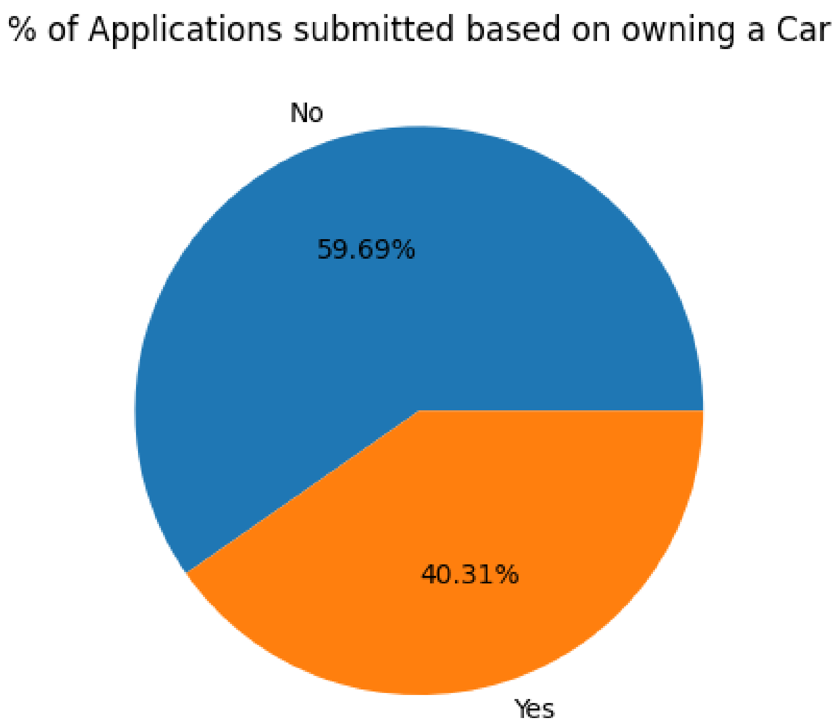
[159]:
```
# This graph shows that, majority of application are submitted by Female's
plt.pie(join_data['GENDER'].value_counts(), labels=['Female', 'Male'],␣
 ↪autopct='%1.2f%%')
plt.title('% of Applications submitted based on Gender')
plt.show()
```



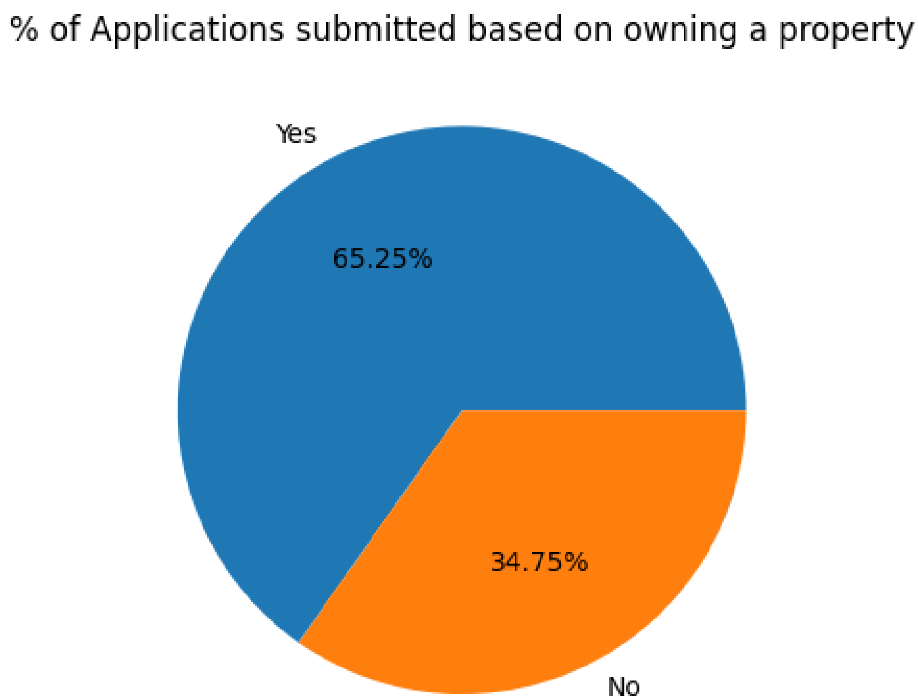
[160]:
```
# This graph shows that, majority of application are approved for Female's
plt.pie(join_data[join_data['label']==0]['GENDER'].value_counts(),␣
 ↪labels=['Female', 'Male'], autopct='%1.2f%%')
plt.title('% of Applications Approved based on Gender')
plt.show()
```



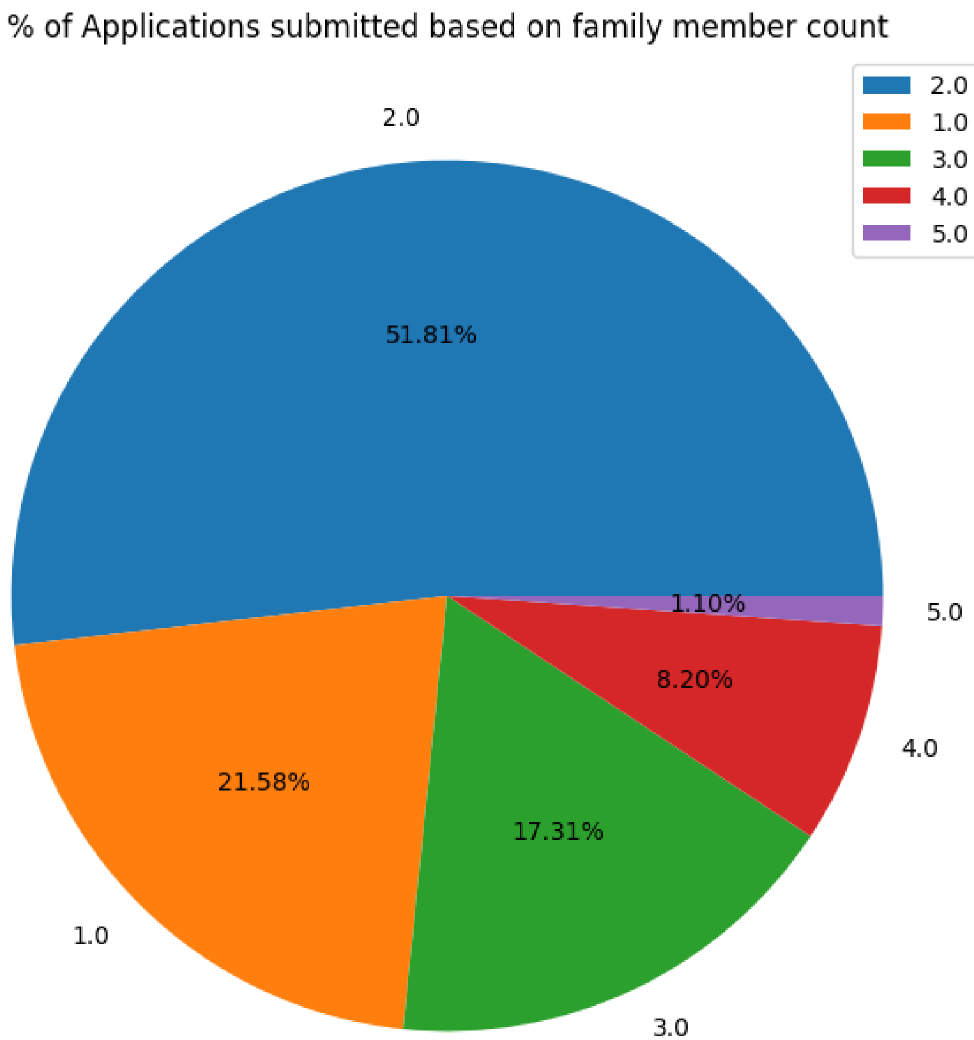
[161]:
```
# This graph shows that, majority of applicatant's don't own a car
plt.pie(join_data['Car_Owner'].value_counts(), labels=['No', 'Yes'],␣
↪autopct='%1.2f%%')
plt.title('% of Applications submitted based on owning a Car')
plt.show()
```



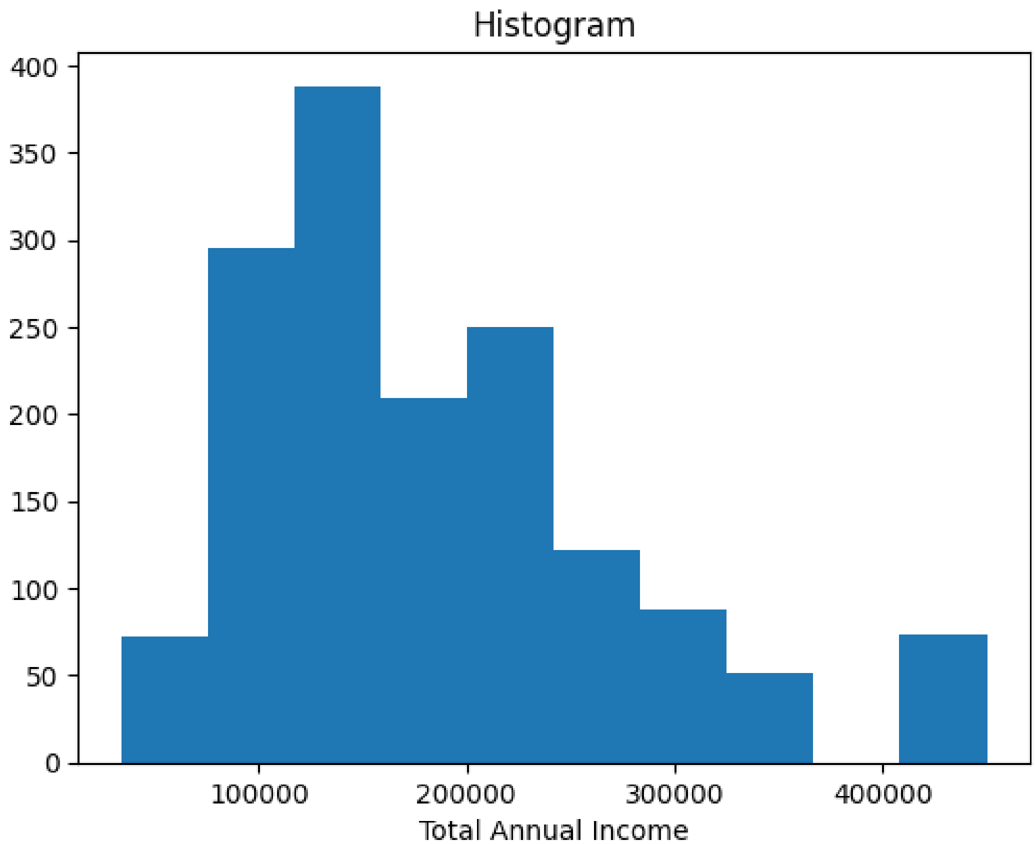
[162]:
```
# This graph shows that, majority of applicatant's own a property / House
plt.pie(join_data['Propert_Owner'].value_counts(), labels=['Yes','No'],␣
↪autopct='%1.2f%%')
plt.title('% of Applications submitted based on owning a property')
plt.show()
```



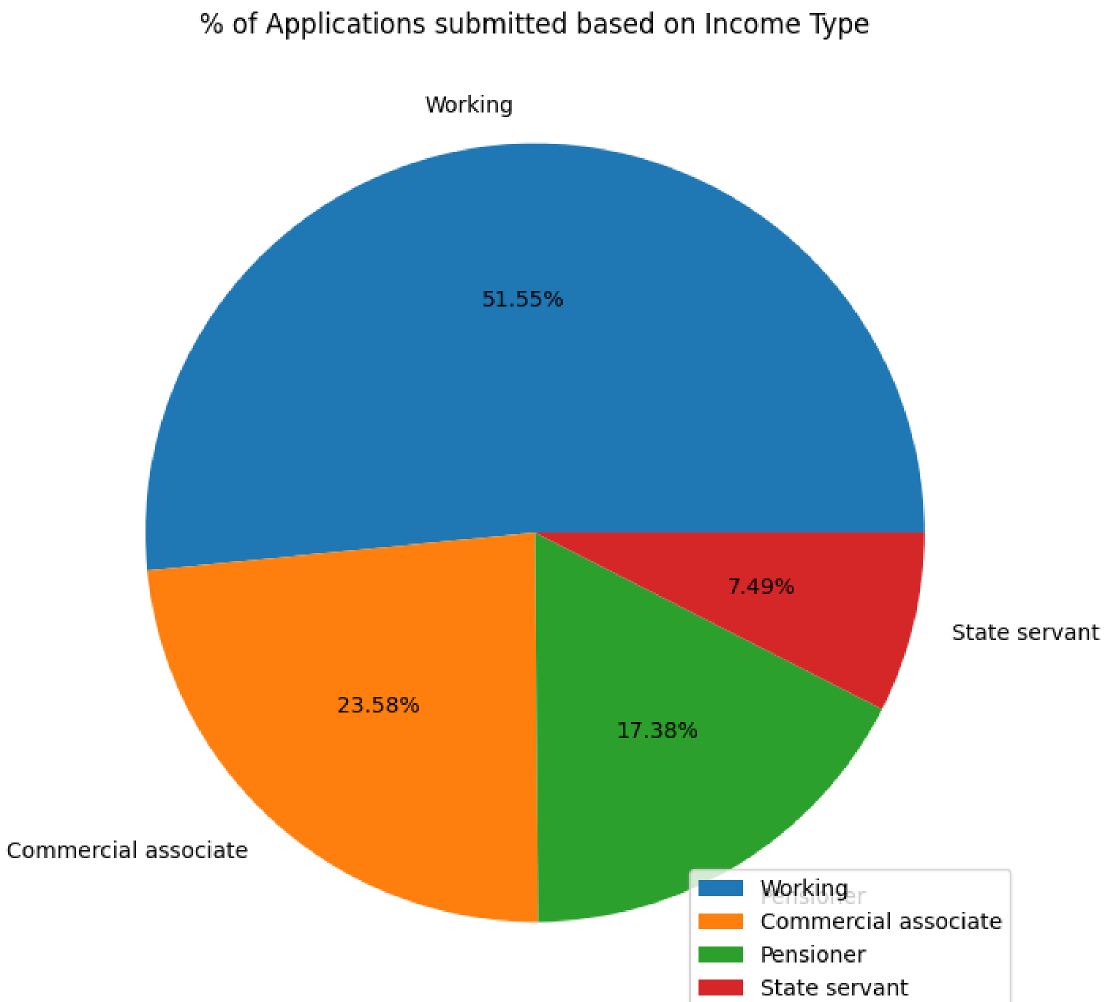
[163]:
```
# This graph shows that, majority of applicatant's have 2 members in family
plt.figure(figsize = (8,8))
plt.pie(join_data['Family_Members'].value_counts(),␣
 ↪labels=join_data['Family_Members'].value_counts().index, autopct='%1.2f%%')
plt.title('% of Applications submitted based on family member count')
plt.legend()
plt.show()
```



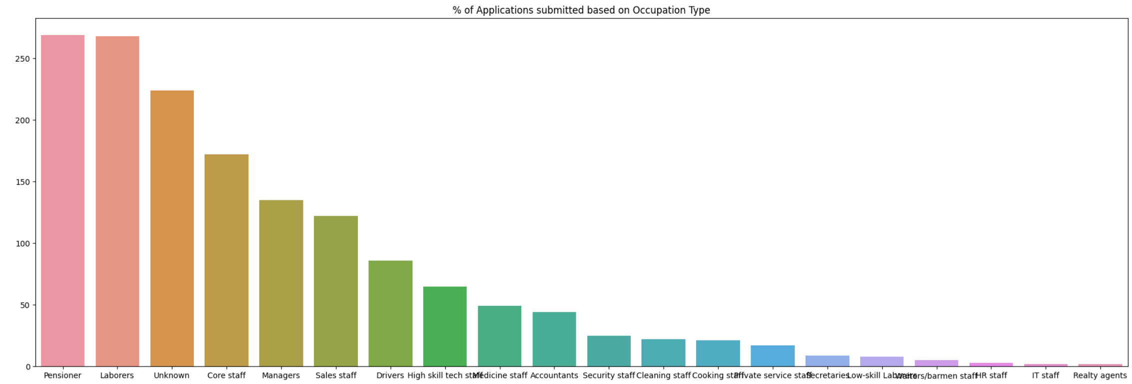
[164]:
```
# This graph shows that, majority of applicatant's income lies between 50k to␣
↪25 lakh
plt.hist(join_data['Annual_income'], bins=10)
plt.xlabel('Total Annual Income')
plt.title('Histogram')
plt.show()
```



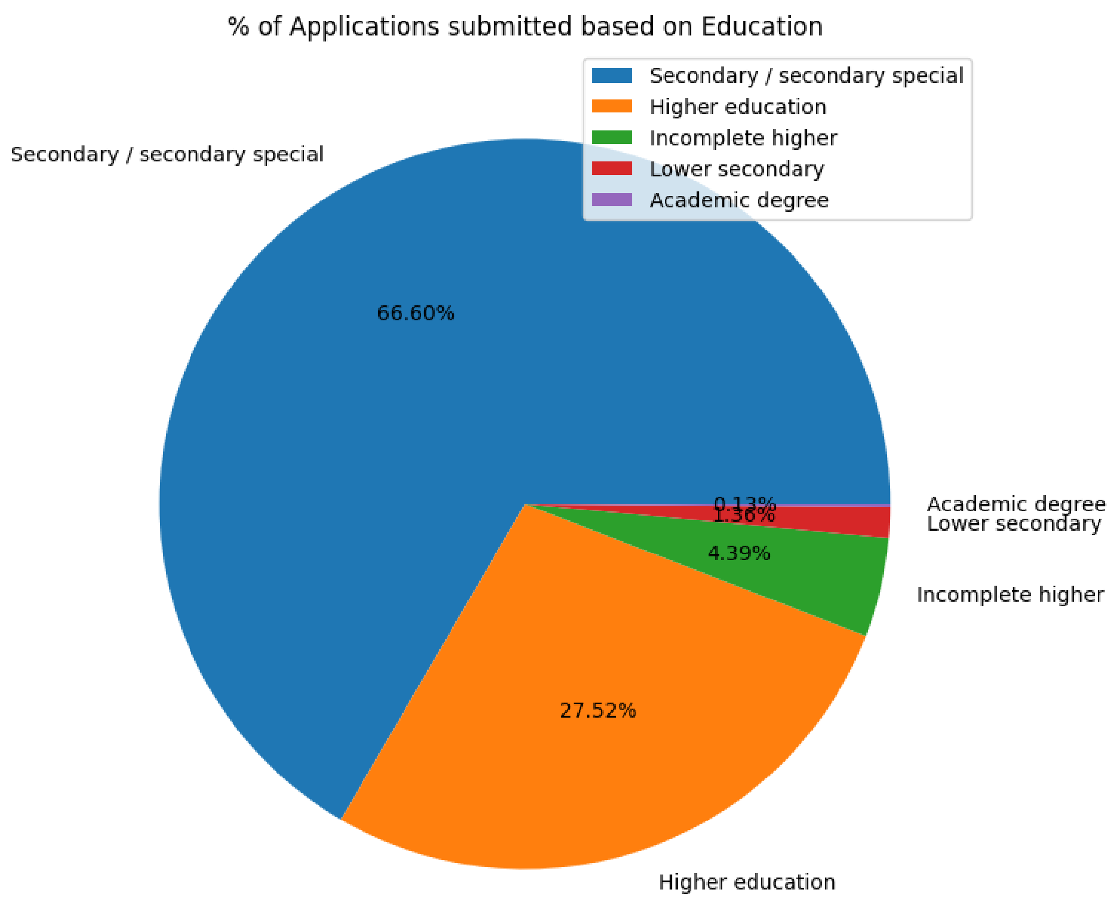
[165]:
```
# This graph shows that, majority of applicatant's are working professional
plt.figure(figsize = (8,8))
plt.pie(join_data['Type_Income'].value_counts(),␣
↪labels=join_data['Type_Income'].value_counts().index, autopct='%1.2f%%')
plt.title('% of Applications submitted based on Income Type')
plt.legend()
plt.show()
```



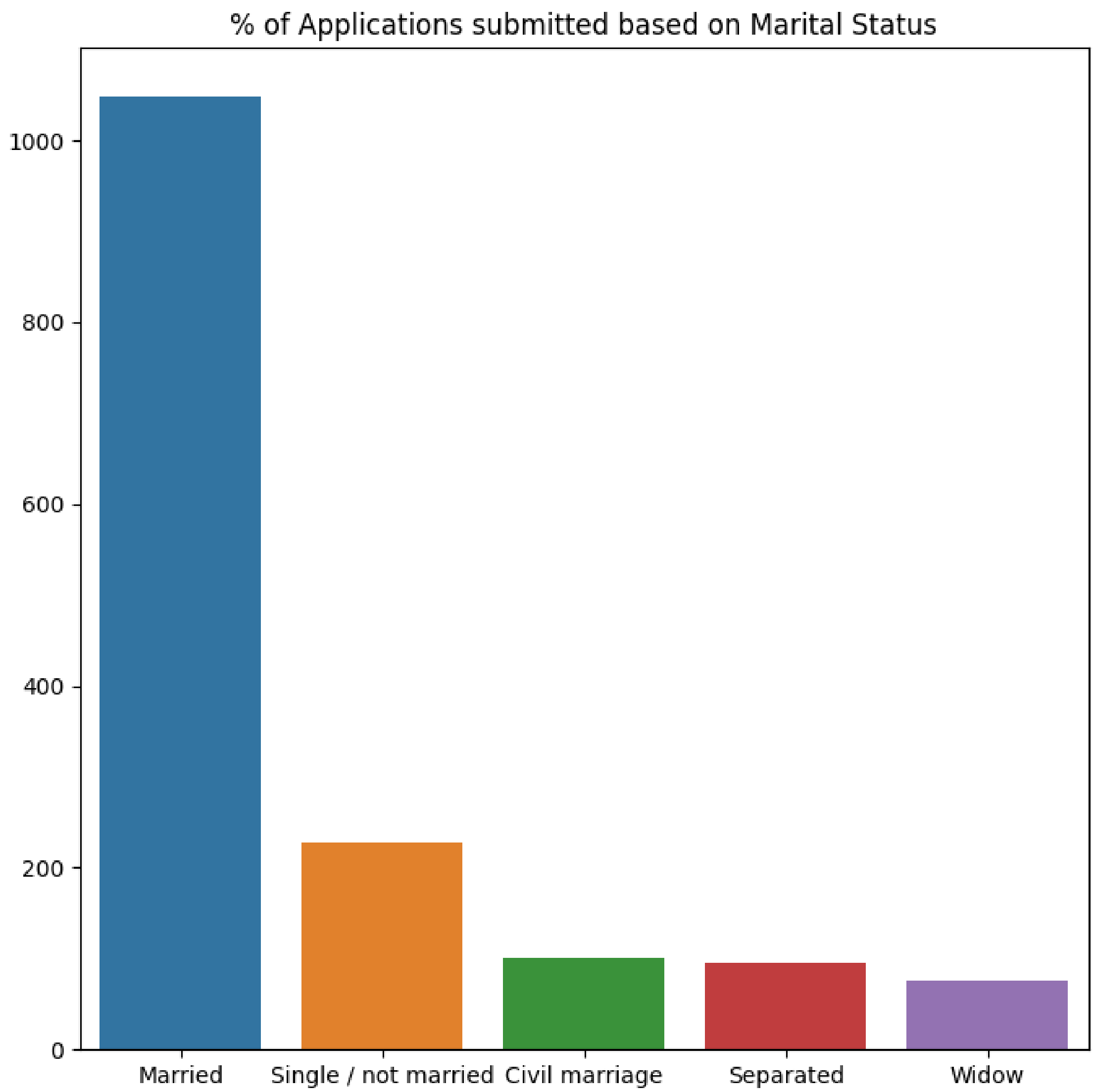
[166]:
```
# This graph shows that the majority of applicants are pensioner and laborers
plt.figure(figsize=(25,8))
sns.barplot(x=join_data['Type_Occupation'].value_counts().index,␣
↪y=join_data['Type_Occupation'].value_counts().values)
plt.title('% of Applications submitted based on Occupation Type')
plt.show()
```



[167]:
```
# This graph shows that, majority of applicatant's completed the Secondary␣
↪Education
plt.figure(figsize=(8,8))
plt.pie(join_data['EDUCATION'].value_counts(), labels=join_data['EDUCATION'].
↪value_counts().index, autopct='%1.2f%%')
plt.title('% of Applications submitted based on Education')
plt.legend()
plt.show()
```

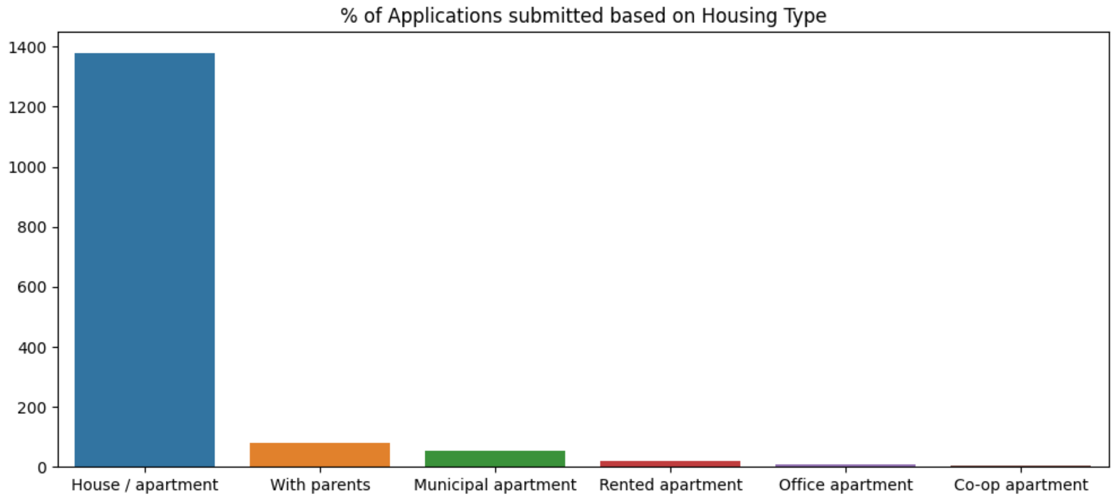



```
[168]: # This graph shows that the majority of applicants are married
plt.figure(figsize=(8, 8))
sns.barplot(x=join_data['Marital_status'].value_counts().index,␣
 ↪y=join_data['Marital_status'].value_counts().values)
plt.title('% of Applications submitted based on Marital Status')
plt.show()
```

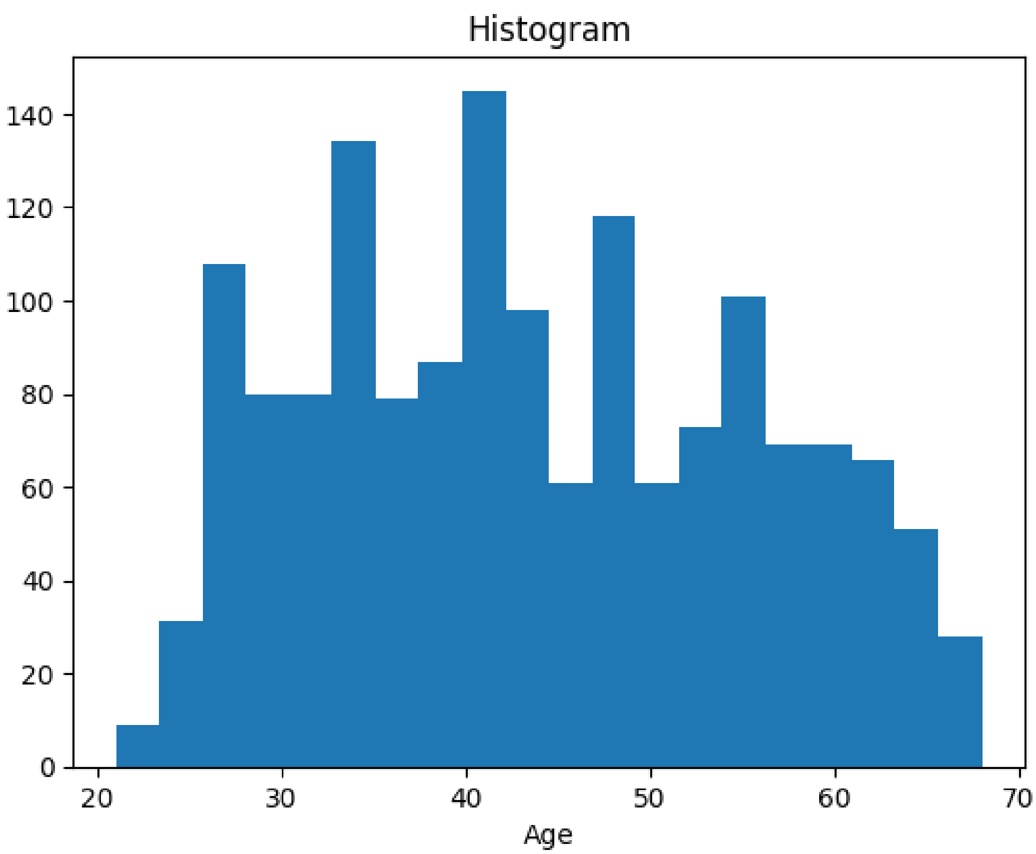


[169]:

```
# This graph shows that, majority of applicatant's lives in House / Apartment
plt.figure(figsize=(12,5))
sns.barplot(x=join_data['Housing_type'].value_counts().index,␣
 ↪y=join_data['Housing_type'].value_counts().values)
plt.title('% of Applications submitted based on Housing Type')
plt.show()
```

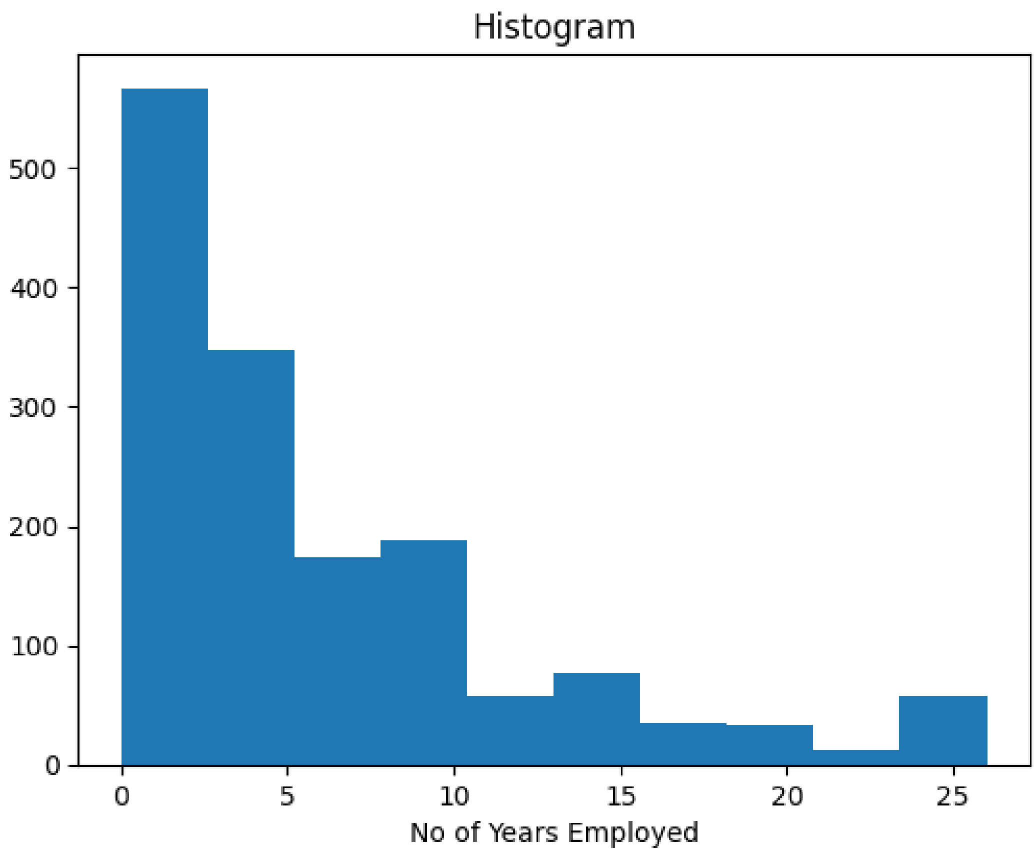


[170]:
```
# This graph shows that, majority of applicatant's are 25 to 65 years old
plt.hist(join_data['AGE'], bins=20)
plt.xlabel('Age')
plt.title('Histogram')
plt.show()
```



[171]:

```
# This graph shows that, majority of applicatant's are Employed for 0 to 10
↪years
plt.hist(join_data['YEAR_EMPLOYED'], bins=10)
plt.xlabel('No of Years Employed')
plt.title('Histogram')
plt.show()
```



[172]:

```
join_data.head()
```

[172]:

```
    Ind_ID GENDER Car_Owner Propert_Owner  Annual_income  \
0  5008827      M         Y             Y       180000.0
1  5009744      F         Y             N       315000.0
2  5009746      F         Y             N       315000.0
3  5009749      F         Y             N       195394.0
4  5009752      F         Y             N       315000.0

            Type_Income         EDUCATION Marital_status       Housing_type  \
```

```
0             Pensioner  Higher education        Married  House / apartment
1  Commercial associate  Higher education        Married  House / apartment
2  Commercial associate  Higher education        Married  House / apartment
3  Commercial associate  Higher education        Married  House / apartment
4  Commercial associate  Higher education        Married  House / apartment

    AGE  YEAR_EMPLOYED Type_Occupation  Family_Members  label
0  51.0            0.0       Pensioner             2.0      1
1  37.0            2.0         Unknown             2.0      1
2  44.0            2.0         Unknown             2.0      1
3  37.0            2.0         Unknown             2.0      1
4  37.0            2.0         Unknown             2.0      1
```

###Encoding

[173]:
```
cat_columns = join_data.columns[(join_data.dtypes =='object').values].tolist()
cat_columns
```

[173]:
```
['GENDER',
 'Car_Owner',
 'Propert_Owner',
 'Type_Income',
 'EDUCATION',
 'Marital_status',
 'Housing_type',
 'Type_Occupation']
```

[174]:
```
#Converting all Non-Numerical Columns to Numerical
from sklearn.preprocessing import LabelEncoder

for col in cat_columns:
        globals()['LE_{}'.format(col)] = LabelEncoder()
        join_data[col] = globals()['LE_{}'.format(col)].
 ↪fit_transform(join_data[col])
join_data.head()
```

[174]:
```
    Ind_ID  GENDER  Car_Owner  Propert_Owner  Annual_income  Type_Income  \
0  5008827       1          1              1       180000.0            1
1  5009744       0          1              0       315000.0            0
2  5009746       0          1              0       315000.0            0
3  5009749       0          1              0       195394.0            0
4  5009752       0          1              0       315000.0            0

   EDUCATION  Marital_status  Housing_type   AGE  YEAR_EMPLOYED  \
0          1               1             1  51.0            0.0
1          1               1             1  37.0            2.0
2          1               1             1  44.0            2.0
```

```
3          1               1             1  37.0            2.0
4          1               1             1  37.0            2.0

   Type_Occupation  Family_Members  label
0               12             2.0      1
1               18             2.0      1
2               18             2.0      1
3               18             2.0      1
4               18             2.0      1
```

[175]:
```
for col in cat_columns:
    print(col , "  : ", globals()['LE_{}'.format(col)].classes_)
```

```
GENDER   :  ['F' 'M']
Car_Owner   :  ['N' 'Y']
Propert_Owner   :  ['N' 'Y']
Type_Income   :  ['Commercial associate' 'Pensioner' 'State servant' 'Working']
EDUCATION   :  ['Academic degree' 'Higher education' 'Incomplete higher'
 'Lower secondary' 'Secondary / secondary special']
Marital_status   :  ['Civil marriage' 'Married' 'Separated' 'Single / not
married' 'Widow']
Housing_type   :  ['Co-op apartment' 'House / apartment' 'Municipal apartment'
 'Office apartment' 'Rented apartment' 'With parents']
Type_Occupation   :  ['Accountants' 'Cleaning staff' 'Cooking staff' 'Core
staff' 'Drivers'
 'HR staff' 'High skill tech staff' 'IT staff' 'Laborers'
 'Low-skill Laborers' 'Managers' 'Medicine staff' 'Pensioner'
 'Private service staff' 'Realty agents' 'Sales staff' 'Secretaries'
 'Security staff' 'Unknown' 'Waiters/barmen staff']
```

[176]:
```
join_data.corr()
```

[176]:
```
                   Ind_ID    GENDER  Car_Owner  Propert_Owner  Annual_income  \
Ind_ID           1.000000  0.027597  -0.046811      -0.050421       0.007804
GENDER           0.027597  1.000000   0.366257      -0.038264       0.256974
Car_Owner       -0.046811  0.366257   1.000000       0.002401       0.234180
Propert_Owner   -0.050421 -0.038264   0.002401       1.000000       0.024442
Annual_income    0.007804  0.256974   0.234180       0.024442       1.000000
Type_Income      0.026832  0.061954   0.033180      -0.057481      -0.115998
EDUCATION        0.020761 -0.034325  -0.131209      -0.018622      -0.257999
Marital_status   0.014426 -0.120783  -0.135318       0.004493      -0.013822
Housing_type     0.024882  0.081154  -0.001358      -0.174783       0.021035
AGE             -0.022025 -0.183135  -0.142936       0.123679      -0.108060
YEAR_EMPLOYED    0.001698 -0.025230  -0.006679      -0.055736       0.066598
Type_Occupation  0.001800 -0.083036  -0.078538       0.004151      -0.048204
Family_Members   0.020478  0.096845   0.119357      -0.010574       0.048890
label            0.016796  0.045664  -0.014734      -0.017906       0.034179
```

```
                 Type_Income  EDUCATION  Marital_status  Housing_type  \
Ind_ID              0.026832   0.020761        0.014426      0.024882
GENDER              0.061954  -0.034325       -0.120783      0.081154
Car_Owner           0.033180  -0.131209       -0.135318     -0.001358
Propert_Owner      -0.057481  -0.018622        0.004493     -0.174783
Annual_income      -0.115998  -0.257999       -0.013822      0.021035
Type_Income         1.000000   0.100511       -0.032925      0.025516
EDUCATION           0.100511   1.000000        0.051966     -0.044552
Marital_status     -0.032925   0.051966        1.000000     -0.009247
Housing_type        0.025516  -0.044552       -0.009247      1.000000
AGE                -0.171505   0.189292        0.115300     -0.218762
YEAR_EMPLOYED       0.184963   0.018455       -0.100822     -0.038571
Type_Occupation    -0.142484   0.007640        0.033525     -0.024982
Family_Members      0.067976  -0.074409       -0.574837      0.004284
label              -0.067856  -0.027040        0.057885     -0.001610

                      AGE  YEAR_EMPLOYED  Type_Occupation  Family_Members  \
Ind_ID          -0.022025       0.001698         0.001800        0.020478
GENDER          -0.183135      -0.025230        -0.083036        0.096845
Car_Owner       -0.142936      -0.006679        -0.078538        0.119357
Propert_Owner    0.123679      -0.055736         0.004151       -0.010574
Annual_income   -0.108060       0.066598        -0.048204        0.048890
Type_Income     -0.171505       0.184963        -0.142484        0.067976
EDUCATION        0.189292       0.018455         0.007640       -0.074409
Marital_status   0.115300      -0.100822         0.033525       -0.574837
Housing_type    -0.218762      -0.038571        -0.024982        0.004284
AGE              1.000000      -0.037503         0.102825       -0.283925
YEAR_EMPLOYED   -0.037503       1.000000        -0.117707        0.112905
Type_Occupation  0.102825      -0.117707         1.000000       -0.054058
Family_Members  -0.283925       0.112905        -0.054058        1.000000
label            0.044841      -0.092623        -0.014005       -0.032135

                    label
Ind_ID           0.016796
GENDER           0.045664
Car_Owner       -0.014734
Propert_Owner   -0.017906
Annual_income    0.034179
Type_Income     -0.067856
EDUCATION       -0.027040
Marital_status   0.057885
Housing_type    -0.001610
AGE              0.044841
YEAR_EMPLOYED   -0.092623
Type_Occupation -0.014005
Family_Members  -0.032135
```

```
label            1.000000
```

[177]:
```
features = join_data.drop(['label'], axis=1)
label = join_data['label']
```

[178]:
```
features.head()
```

[178]:
```
    Ind_ID  GENDER  Car_Owner  Propert_Owner  Annual_income  Type_Income  \
0  5008827       1          1              1       180000.0            1
1  5009744       0          1              0       315000.0            0
2  5009746       0          1              0       315000.0            0
3  5009749       0          1              0       195394.0            0
4  5009752       0          1              0       315000.0            0

   EDUCATION  Marital_status  Housing_type   AGE  YEAR_EMPLOYED  \
0          1               1             1  51.0            0.0
1          1               1             1  37.0            2.0
2          1               1             1  44.0            2.0
3          1               1             1  37.0            2.0
4          1               1             1  37.0            2.0

   Type_Occupation  Family_Members
0               12             2.0
1               18             2.0
2               18             2.0
3               18             2.0
4               18             2.0
```

[179]:
```
label.head()
```

[179]:
```
0    1
1    1
2    1
3    1
4    1
Name: label, dtype: int64
```

###Splitting dataset

[180]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(features,label,test_size=0.
 ↪2,random_state = 10)
```

###Feature Scaling

[181]:
```
# scaling all features
from sklearn.preprocessing import MinMaxScaler
MMS = MinMaxScaler()
```

```
x_train_scaled = pd.DataFrame(MMS.fit_transform(x_train), columns=x_train.
 ↪columns)
x_test_scaled = pd.DataFrame(MMS.transform(x_test), columns=x_test.columns)
```

###Balancing dataset

[182]:
```
# adding samples to minority class using SMOTE
from imblearn.over_sampling import SMOTE
oversample = SMOTE()

x_train_oversam, y_train_oversam = oversample.fit_resample(x_train_scaled,␣
 ↪y_train)
x_test_oversam, y_test_oversam = oversample.fit_resample(x_test_scaled, y_test)
```

[183]:
```
# Original majority and minority class
y_train.value_counts(normalize=True)*100
```

[183]:
```
0    88.772213
1    11.227787
Name: label, dtype: float64
```

[184]:
```
# after using SMOTE
y_train_oversam.value_counts(normalize=True)*100
```

[184]:
```
0    50.0
1    50.0
Name: label, dtype: float64
```

##Machine Learning Model

###Logistic Regression

[185]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix

#creating logistic regression object
log_model = LogisticRegression(random_state=0)
#passing independent and dependent training data into the model
log_model.fit(x_train_oversam, y_train_oversam)

print('Logistic Model Accuracy : ', log_model.score(x_test_oversam,␣
 ↪y_test_oversam)*100, '%')

#model to get prediction for test data
prediction = log_model.predict(x_test_oversam)

print('\nConfusion matrix :')
```

```
print(confusion_matrix(y_test_oversam, prediction))

print('\nClassification report:')
print(classification_report(y_test_oversam, prediction))
```

```
Logistic Model Accuracy :  62.22627737226277 %

Confusion matrix :
[[151 123]
 [ 84 190]]

Classification report:
              precision    recall  f1-score   support

           0       0.64      0.55      0.59       274
           1       0.61      0.69      0.65       274

    accuracy                           0.62       548
   macro avg       0.62      0.62      0.62       548
weighted avg       0.62      0.62      0.62       548
```

0.0.1 Decision Tree classification

[186]:
```
from sklearn.tree import DecisionTreeClassifier

#create decision tree classifier object
decision_model = DecisionTreeClassifier(max_depth=13,criterion="entropy")

#train decision tree classifier
decision_model.fit(x_train_oversam, y_train_oversam)

print('Decision Tree Model Accuracy : ', decision_model.score(x_test_oversam,␣
 ↪y_test_oversam)*100, '%')

#predict the response for test dataset
prediction = decision_model.predict(x_test_oversam)

print('\nConfusion matrix :')
print(confusion_matrix(y_test_oversam, prediction))

print('\nClassification report:')
print(classification_report(y_test_oversam, prediction))
```

```
Decision Tree Model Accuracy :  78.83211678832117 %

Confusion matrix :
[[232  42]
```

```
 [ 74 200]]

Classification report:
              precision    recall  f1-score   support

           0       0.76      0.85      0.80       274
           1       0.83      0.73      0.78       274

    accuracy                           0.79       548
   macro avg       0.79      0.79      0.79       548
weighted avg       0.79      0.79      0.79       548
```

###Random Forest classification

[187]:
```python
from sklearn.ensemble import RandomForestClassifier

RandomForest_model = 
 ↪RandomForestClassifier(n_estimators=9,max_depth=13,criterion="entropy")

RandomForest_model.fit(x_train_oversam, y_train_oversam)

print('Random Forest Model Accuracy : ', RandomForest_model.
 ↪score(x_test_oversam, y_test_oversam)*100, '%')

prediction = RandomForest_model.predict(x_test_oversam)
print('\nConfusion matrix :')
print(confusion_matrix(y_test_oversam, prediction))

print('\nClassification report:')
print(classification_report(y_test_oversam, prediction))
```

```
Random Forest Model Accuracy :  81.2043795620438 %

Confusion matrix :
[[253  21]
 [ 82 192]]

Classification report:
              precision    recall  f1-score   support

           0       0.76      0.92      0.83       274
           1       0.90      0.70      0.79       274

    accuracy                           0.81       548
   macro avg       0.83      0.81      0.81       548
weighted avg       0.83      0.81      0.81       548
```

###Support Vector Machine classification

```
[188]: from sklearn.svm import SVC

svc_model =SVC()

svc_model.fit(x_train_oversam, y_train_oversam)

print('Support Vector Classifier Accuracy : ', svc_model.score(x_test_oversam,␣
 ↪y_test_oversam)*100, '%')

prediction = svc_model.predict(x_test_oversam)
print('\nConfusion matrix :')
print(confusion_matrix(y_test_oversam, prediction))

print('\nClassification report:')
print(classification_report(y_test_oversam, prediction))
```

```
Support Vector Classifier Accuracy :  72.08029197080292 %

Confusion matrix :
[[206  68]
 [ 85 189]]

Classification report:
              precision    recall  f1-score   support

           0       0.71      0.75      0.73       274
           1       0.74      0.69      0.71       274

    accuracy                           0.72       548
   macro avg       0.72      0.72      0.72       548
weighted avg       0.72      0.72      0.72       548
```

###K Nearest Neighbor classification

```
[189]: from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors =3)

knn_model.fit(x_train_oversam, y_train_oversam)

print('KNN Model Accuracy : ', knn_model.score(x_test_oversam,␣
 ↪y_test_oversam)*100, '%')

prediction = knn_model.predict(x_test_oversam)
print('\nConfusion matrix :')
```

```python
print(confusion_matrix(y_test_oversam, prediction))

print('\nClassification report:')
print(classification_report(y_test_oversam, prediction))
```

```
KNN Model Accuracy :  71.35036496350365 %

Confusion matrix :
[[218  56]
 [101 173]]

Classification report:
              precision    recall  f1-score   support

           0       0.68      0.80      0.74       274
           1       0.76      0.63      0.69       274

    accuracy                           0.71       548
   macro avg       0.72      0.71      0.71       548
weighted avg       0.72      0.71      0.71       548
```

###XGBoost classification

[190]:
```python
from xgboost import XGBClassifier

XGB_model = XGBClassifier()

XGB_model.fit(x_train_oversam, y_train_oversam)

print('XGBoost Model Accuracy : ', XGB_model.score(x_test_oversam,␣
 ↪y_test_oversam)*100, '%')

prediction = XGB_model.predict(x_test_oversam)
print('\nConfusion matrix :')
print(confusion_matrix(y_test_oversam, prediction))

print('\nClassification report:')
print(classification_report(y_test_oversam, prediction))
```

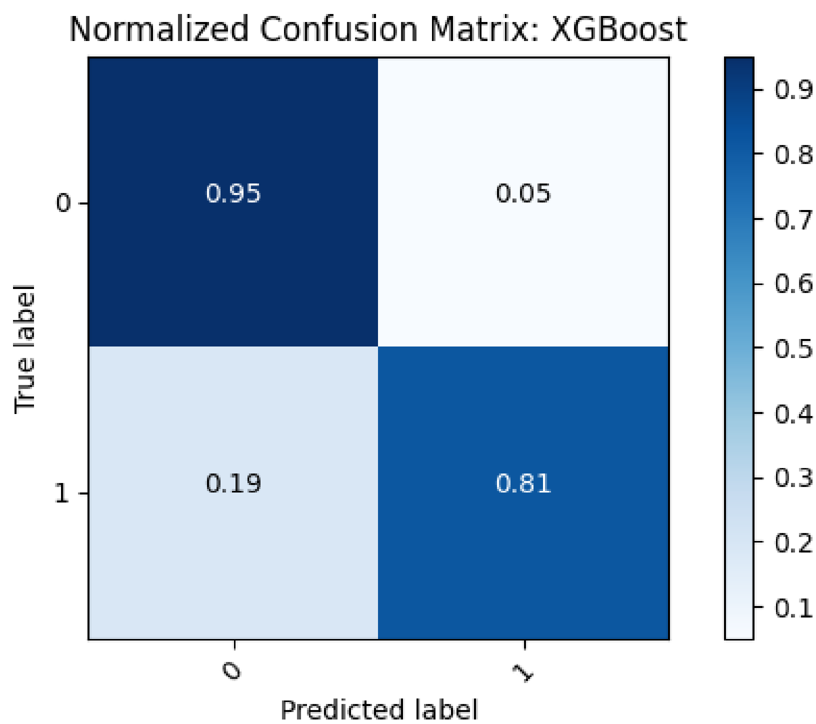
```
XGBoost Model Accuracy :  88.13868613138686 %

Confusion matrix :
[[260  14]
 [ 51 223]]

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

```
           0       0.84      0.95      0.89       274
           1       0.94      0.81      0.87       274

    accuracy                           0.88       548
   macro avg       0.89      0.88      0.88       548
weighted avg       0.89      0.88      0.88       548
```

[191]:

```
cm = confusion_matrix(y_test_oversam, prediction)

# Class names
class_names = np.unique(y_test_oversam)

def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion␣
↪Matrix', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.figure(figsize=(6,4))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(len(classes)):
        for j in range(len(classes)):
            plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center",␣
↪color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Plot normalized confusion matrix
plot_confusion_matrix(cm, classes=class_names, normalize=True,␣
↪title='Normalized Confusion Matrix: XGBoost')
plt.show()
```

```
Normalized confusion matrix
```



##Validation

###K-Fold Cross Validation

[192]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
kfold = KFold(5)
```

####Logistic Regression

[200]:
```
results=cross_val_score(log_model,features,label,cv=kfold)
accuracy_score=results*100
print('accuracy of each fold - {}'.format(accuracy_score))

avg_accuracy_score=np.mean(results)*100
print('Avg accuracy : {}'.format(avg_accuracy_score))
```

```
accuracy of each fold - [88.70967742 88.70967742 88.70967742 88.67313916
88.67313916]
Avg accuracy : 88.69506211504333
```

Decision Tree classification

[194]:
```
results=cross_val_score(decision_model,features,label,cv=kfold)
accuracy_score=results*100
```

```
print('accuracy of each fold - {}'.format(accuracy_score))

avg_accuracy_score=np.mean(results)*100
print('Avg accuracy : {}'.format(avg_accuracy_score))
```

```
accuracy of each fold - [81.61290323 79.35483871 81.61290323 82.20064725
82.84789644]
Avg accuracy : 81.52583777012214
```

####Random Forest classification

[195]:
```
results=cross_val_score(RandomForest_model,features,label,cv=kfold)
accuracy_score=results*100
print('accuracy of each fold - {}'.format(accuracy_score))

avg_accuracy_score=np.mean(results)*100
print('Avg accuracy : {}'.format(avg_accuracy_score))
```

```
accuracy of each fold - [88.38709677 88.70967742 87.09677419 87.70226537
87.70226537]
Avg accuracy : 87.91961582628667
```

Support Vector Machine classification

[199]:
```
results=cross_val_score(svc_model,features,label,cv=kfold)
accuracy_score=results*100
print('accuracy of each fold - {}'.format(accuracy_score))

avg_accuracy_score=np.mean(results)*100
print('Avg accuracy : {}'.format(avg_accuracy_score))
```

```
accuracy of each fold - [88.70967742 88.70967742 88.70967742 88.67313916
88.67313916]
Avg accuracy : 88.69506211504333
```

####K Nearest Neighbor classification

[197]:
```
results=cross_val_score(knn_model,features,label,cv=kfold)
accuracy_score=results*100
print('accuracy of each fold - {}'.format(accuracy_score))

avg_accuracy_score=np.mean(results)*100
print('Avg accuracy : {}'.format(avg_accuracy_score))
```

```
accuracy of each fold - [82.90322581 84.51612903 83.87096774 83.81877023
85.7605178 ]
Avg accuracy : 84.17392212130703
```

####XGBoost classification

[198]:

```
results=cross_val_score(XGB_model,features,label,cv=kfold)
accuracy_score=results*100
print('accuracy of each fold - {}'.format(accuracy_score))

avg_accuracy_score=np.mean(results)*100
print('Avg accuracy : {}'.format(avg_accuracy_score))
```

```
accuracy of each fold - [86.77419355 86.77419355 87.41935484 86.40776699
87.05501618]
Avg accuracy : 86.886105021401
```