final-credit-card-prediction

April 10, 2023

#INTRODUCTION

###Credit card:- #####A credit card is a thin rectangular piece of plastic or metal issued by a bank or financial services company that allows cardholders to borrow funds with which to pay for goods and services with merchants that accept cards for payment. Credit cards impose the condition that cardholders pay back the borrowed money, plus any applicable interest, as well as any additional agreed-upon charges, either in full by the billing date or over time.

###Advantages of credit card :- * Earn rewards such as cash back or miles points.

- Protection against credit card fraud.
- Credit score information for free.
- No foreign transaction fees.
- Increased purchasing power.
- Not linked to checking or savings account.
- Putting a hold on a rental car or hotel room.

###Eligibilty for creditcard :- * Age * Income * Residency * Citizenship

- 1) Why is your proposal important in today's world? How predicting a good client is worthy for a bank?
  - A) In today's world, where online transactions have become the norm, credit card fraud has become a major concern for banks and financial institutions. It is essential for banks to accurately predict whether a client is a good or bad credit risk to minimize the risk of financial losses from default or fraud. This is where credit card approval prediction comes in.
- 2) How is it going to impact the banking sector?
  - By analyzing a customer's credit history, income, employment status, and other relevant data, banks can predict the likelihood of a customer defaulting on their credit card payments. This can help the bank make informed decisions on whether or not to approve a credit card application and what credit limit to assign.
- 3) What is the gap in the knowledge or how you proposed method can be helpful?
  - Predicting a good client is important for a bank for several reasons.

– It helps the bank minimize the risk of financial loss due to default or fraud. *It helps the bank determine an appropriate credit limit for the customer, which can help prevent the customer from accumulating debt they can't repay.
– It helps the bank identify potential high-value customers who are likely to spend more on their credit cards and generate more revenue for the bank.

#GATHERING DATA

[1]:
```
import pandas as pd   #Library for manipualtion and filtering the data.
import numpy as np  #Library for scientific computing.
import matplotlib.pyplot as plt  #Library for Data visualization.
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

[2]:
```
#Importing Credit Card dataset using read_csv in pandas.
df_credit_card = pd.read_csv("Credit_card.csv")

#Dataset of Credit Card has 1548 'records' and 18 'features'.
df_credit_card.shape
```

[2]: (1548, 18)

[3]:
```
#Importing Credit Card Label dataset using read_csv in pandas.
df_label = pd.read_csv("Credit_card_label.csv")

#Dataset of Credit Card labels has 1548 'records' and 2 'features'.
df_label.shape
```

[3]: (1548, 2)

[4]:
```
#Combining common datset using merge on "Ind_ID" column.
df = pd.merge(df_credit_card, df_label, on='Ind_ID', how='inner')

#Dataset of Credit Card 1548 'records' and 19 'features'.
df.shape
```

[4]: (1548, 19)

##BASIC EXPLORATION

###Columns description:- FIRST FILE (Credit_card.csv) * Ind_ID: Client ID

- Gender : Gender information
- Car_owner : Having car or not
- Propert_owner : Having property or not
- Children : Count of children

- Annual_income : Annual income
- Type_Income : Income type
- Education : Education level
- Marital_status : Marital_status
- Housing_type : Living style
- Birthday_count : Use backward count from current day (0), -1 means yesterday.
- Employed_days : Start date of employment. Use backward count from current day (0). Positive value means, individual is currently unemployed.
- Mobile_phone : Any mobile phone
- Work_phone : Any work phone
- Phone : Any phone number
- EMAIL_ID : Any email ID
- Type_Occupation : Occupation
- Family_Members : Family size

SECOND FILE (Credit_card_label.csv)

- ID : The joining key between application data and credit status data, same is Ind_ID
- Label : 0 is application approved and 1 is application rejected.

###Data Pre-Processing

[5]:
```
#"head()" used to get first 5 rows of dataset.
df.head()
```

[5]:
```
    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
```

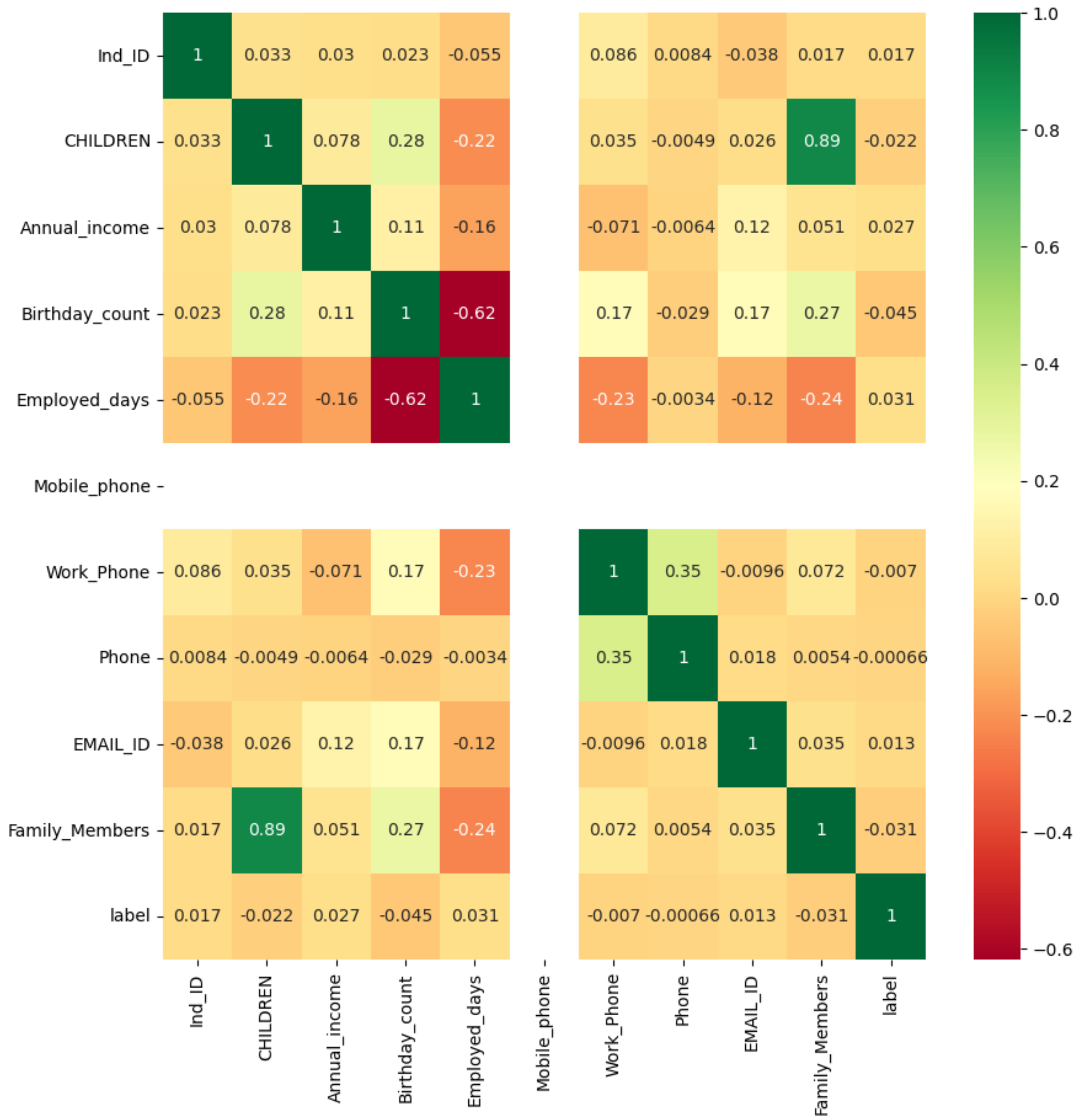
```
2              NaN           -586              1             1       1          0
3         -13557.0           -586              1             1       1          0
4         -13557.0           -586              1             1       1          0

  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
```

[6]:
```
#"describe()" used to give statistics for numerical parameteric column.
df.describe()
```

[6]:
```
             Ind_ID     CHILDREN  Annual_income  Birthday_count  \
count  1.548000e+03  1548.000000   1.525000e+03     1526.000000
mean   5.078920e+06     0.412791   1.913993e+05   -16040.342071
std    4.171759e+04     0.776691   1.132530e+05     4229.503202
min    5.008827e+06     0.000000   3.375000e+04   -24946.000000
25%    5.045070e+06     0.000000   1.215000e+05   -19553.000000
50%    5.078842e+06     0.000000   1.665000e+05   -15661.500000
75%    5.115673e+06     1.000000   2.250000e+05   -12417.000000
max    5.150412e+06    14.000000   1.575000e+06    -7705.000000

       Employed_days  Mobile_phone   Work_Phone        Phone     EMAIL_ID  \
count    1548.000000        1548.0  1548.000000  1548.000000  1548.000000
mean    59364.689922           1.0     0.208010     0.309432     0.092377
std    137808.062701           0.0     0.406015     0.462409     0.289651
min    -14887.000000           1.0     0.000000     0.000000     0.000000
25%     -3174.500000           1.0     0.000000     0.000000     0.000000
50%     -1565.000000           1.0     0.000000     0.000000     0.000000
75%      -431.750000           1.0     0.000000     1.000000     0.000000
max    365243.000000           1.0     1.000000     1.000000     1.000000

       Family_Members        label
count     1548.000000  1548.000000
mean         2.161499     0.113049
std          0.947772     0.316755
min          1.000000     0.000000
25%          2.000000     0.000000
50%          2.000000     0.000000
75%          3.000000     0.000000
max         15.000000     1.000000
```

[7]:
```
#Defining the correlation between the attributes.
corr_df = df.corr()
top_corr_fig = corr_df.index
```

```
plt.figure(figsize=(10,10))
sns.heatmap(df[top_corr_fig].corr(), annot=True, cmap ="RdYlGn")
plt.show()
```



Observation :-

- Numerical features - Annual Income, Birthday count, Employeed Days, Family Members and Children are more likely correlated to Credit Card Approval.
- Birthday count and Family Members are inversely correlated to “Label”.
- Other are directly correlated.

[8]:
```
#"nunique()" used to distinct values in each column in dataset.
df.nunique()
```

[8]:
```
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
label                 2
dtype: int64
```

[9]:
```
#Dataset of Credit Card has 1548 rows and 19 columns.
df.shape
```

[9]:
```
(1548, 19)
```

There are 1548 Samples with 19 featues/attributes.

[10]:
```
#"isnull().sum()" give total null values in each column in dataset.
df.isnull().sum()
```

[10]:
```
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
```

[11]:
```
#"isnull().sum().sum()" gives total null values.
df.isnull().sum().sum()
```

[11]:
```
540
```

There are total 540 Missing values :-

(May be)

- MCAR(Missing Completely At Random) - Data not collected accordingly.
- MAR(Missing At Random) - Data not filled properly by customers.
- MNAR(Missing Not At Random) - Data not filled intensionly.

[12]:
```
#Missing value percent
df.isnull().mean()*100
```

[12]:
```
Ind_ID              0.000000
GENDER              0.452196
Car_Owner           0.000000
Propert_Owner       0.000000
CHILDREN            0.000000
Annual_income       1.485788
Type_Income         0.000000
EDUCATION           0.000000
Marital_status      0.000000
Housing_type        0.000000
Birthday_count      1.421189
Employed_days       0.000000
Mobile_phone        0.000000
Work_Phone          0.000000
Phone               0.000000
EMAIL_ID            0.000000
Type_Occupation    31.524548
Family_Members      0.000000
label               0.000000
dtype: float64
```

Observations :- * There are missing values in 4 features :- * Gender = 0.45% * Annual Income = 1.48% * Age = 1.42% * Type of Occupation = 31.5%

[13]:
```python
#Extracting out the values count in columns using loop and "value_counts()".
column_value_count = ["GENDER", "Car_Owner", "Propert_Owner", "CHILDREN",
 ↪"Type_Income", "EDUCATION", "Marital_status", "Housing_type",
 ↪"Mobile_phone", "Work_Phone", "Phone", "EMAIL_ID", "Type_Occupation",
 ↪"Family_Members", "label"]
for i in df[column_value_count]:
  print(df[i].value_counts())
  print("\n--------------------------------------\n")
```

```
F    973
M    568
Name: GENDER, dtype: int64

--------------------------------------


N    924
Y    624
Name: Car_Owner, dtype: int64

--------------------------------------


Y    1010
N     538
Name: Propert_Owner, dtype: int64

--------------------------------------


0     1091
1      305
2      134
3       16
4        1
14       1
Name: CHILDREN, dtype: int64

--------------------------------------


Working                 798
Commercial associate    365
Pensioner               269
State servant           116
Name: Type_Income, dtype: int64

--------------------------------------


Secondary / secondary special    1031
Higher education                  426
```

```
Incomplete higher                  68
Lower secondary                    21
Academic degree                     2
Name: EDUCATION, dtype: int64

--------------------------------------

Married                 1049
Single / not married     227
Civil marriage           101
Separated                 96
Widow                     75
Name: Marital_status, dtype: int64

--------------------------------------

House / apartment      1380
With parents             80
Municipal apartment      53
Rented apartment         21
Office apartment          9
Co-op apartment           5
Name: Housing_type, dtype: int64

--------------------------------------

1    1548
Name: Mobile_phone, dtype: int64

--------------------------------------

0    1226
1     322
Name: Work_Phone, dtype: int64

--------------------------------------

0    1069
1     479
Name: Phone, dtype: int64

--------------------------------------

0    1405
1     143
Name: EMAIL_ID, dtype: int64

--------------------------------------
```

```
Laborers                 268
Core staff               174
Managers                 136
Sales staff              122
Drivers                   86
High skill tech staff     65
Medicine staff            50
Accountants               44
Security staff            25
Cleaning staff            22
Cooking staff             21
Private service staff     17
Secretaries                9
Low-skill Laborers         9
Waiters/barmen staff       5
HR staff                   3
IT staff                   2
Realty agents              2
Name: Type_Occupation, dtype: int64

--------------------------------------

2     802
1     334
3     268
4     127
5      15
6       1
15      1
Name: Family_Members, dtype: int64

--------------------------------------

0    1373
1     175
Name: label, dtype: int64

--------------------------------------
```

#**FEATURE ENGINEERING**

###Feature Transformation

Feature transformation is a mathematical transformation in which we apply a mathematical formula to a particular column (feature) and transform the values, which are useful for our further analysis. It is a technique by which we can boost our model performance. It is also known as "Feature Engineering", which creates new features from existing features that may help improve the model performance.

There are 3 type of Feature Transformation techniques :- 1. Function Transformers 2. Power Transformers 3. Quantile Transformers

- **Function Transformation :** Function transformers are the type of feature transformation technique that uses a particular function to transform the data to the normal distribution. Here the particular function is applied to the data observations.

  - In Function Transform, there are following techniques :-

  1. **Log Transform**
  2. **Square Transform**
  3. **Square Root Transform**
  4. **Reciprocal Transform**

- **Power Transformation :** Power Transformation techniques are the type of feature transformation technique where the power is applied to the data observations for transforming the data.

  - In Power Transform, there are following techniques :-
    1. **Box-Cox Transform**
    2. **Yeo-Johnson Transform**

- **Quantile Transformation :** Quantile transformation techniques are the type of feature transformation technique that can be applied to NY numerical data observations. This transformation technique can be implemented using sklearn. In this transformation technique, the input data can be fed to this transformer where this transformer makes the distribution of the output data normal to fed to the further machine learning algorithm.

#####Column : Birthday Count –> Age

- In the dataset, there are some negative values in which needs to be converted.
- In "Employed_days", 0 or positive value should be 0 as "Experience".
- In "Birthday_count", negative value indicates birthday days.
- "Birthday_count" and "Employed_days" columns need to converted into years with positive value in it.

[14]:
```
print(df.shape)
#Extracting out the rows which are greater then 0 in birthday count.
df[df["Birthday_count"]>=0]
```

```
(1548, 19)
```

[14]:
```
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: []
```

[15]:
```
#Renaming the "Birthday_count" to "Age" by dividing 365 to get the Age.
df.rename(columns = {"Birthday_count" : "Age"}, inplace=True)
df["Age"] = round(abs(df["Age"]/365))
# df[df["Employed_days"]>=0].head()
```

#####Column : Employed Days –> Experience

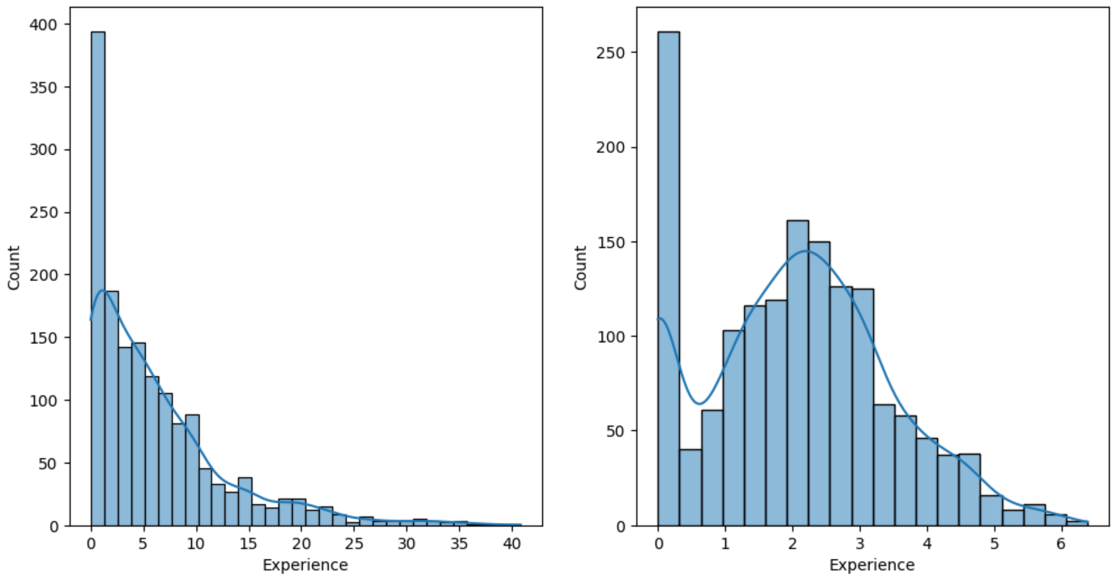
[16]:
```
print(df.shape)
#Extracting out the rows which are greater then 0 in Employed Days.
df["Employed_days"][df["Employed_days"]>=0] = 0
```

```
(1548, 19)
```

[17]:
```
#Renaming the "Employed_days" to "Experience" by dividing 365 to get Experience.
df.rename(columns = {"Employed_days" : "Experience"}, inplace=True)
df["Experience"] = round(abs(df["Experience"]/365),1)
```

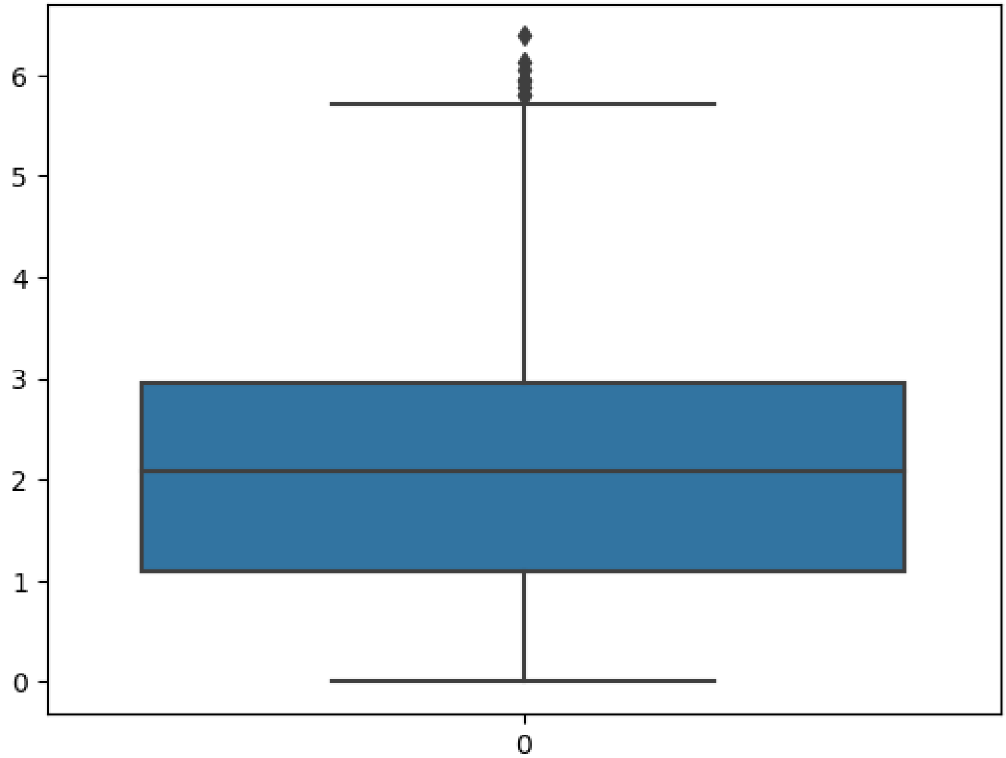
[18]:
```
#Plot of Experience and sqrt(Experience).
plt.figure(figsize=(25,6))
plt.subplot(1, 4, 1)
sns.histplot(df["Experience"], kde = True)
#Before Tranformation
print("Skewness of Experience : ", df["Experience"].skew())

plt.subplot(1, 4, 2)
sns.histplot(np.sqrt(df["Experience"]), kde = True)
print("Skewness of Square root of  Experience : ", np.sqrt(df["Experience"]).
 ↪skew())
plt.show()
```

```
Skewness of Experience :  1.7300609167294572
Skewness of Square root of  Experience :  0.2617332299127787
```



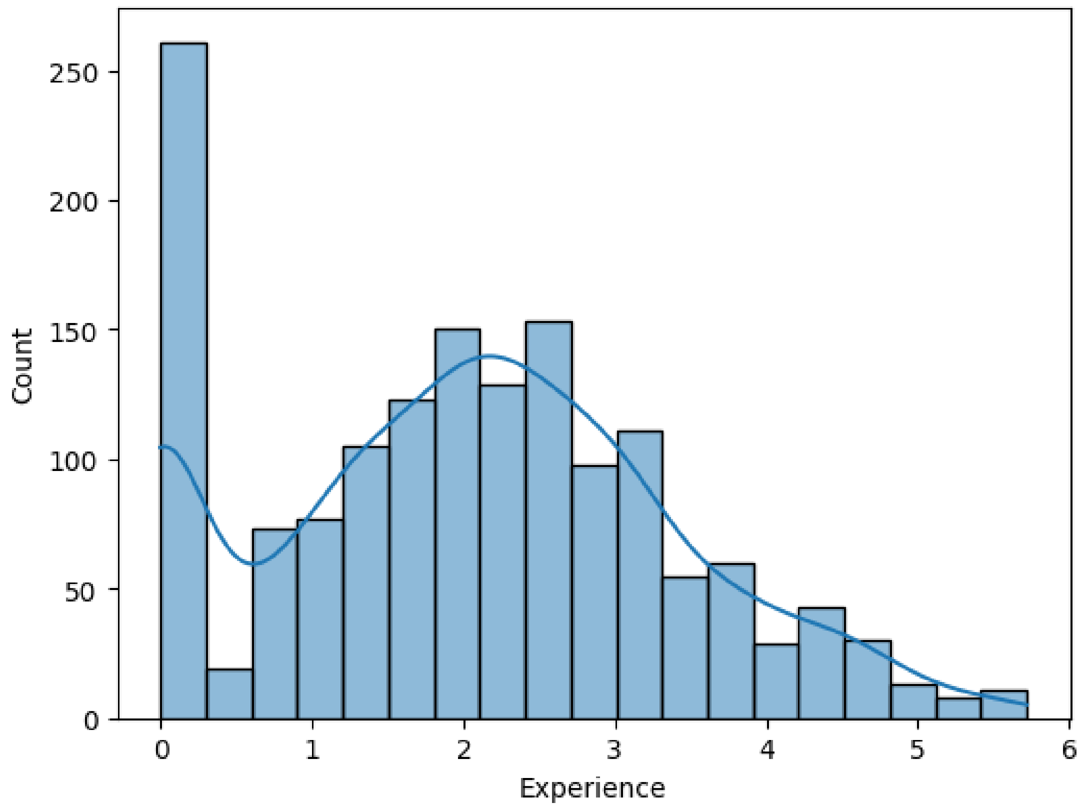

```
[19]: #Replacing the feature with transformed figure that is Square-Root
      ↪Transformation
      df["exp1"] = df['Experience'].copy()
      df["Experience"] = np.sqrt(df["Experience"])
```

```
[20]: #Outliers from the boxplot in plot
      sns.boxplot(df["Experience"])
      plt.show()
```



[21]:
```
#Following the IQR Technique.
q3_exp = df["Experience"].quantile(0.75)
q1_exp = df["Experience"].quantile(0.25)
iqr_exp = q3_exp - q1_exp
upper_exp = q3_exp + (1.5*iqr_exp)
lower_exp = q1_exp - (1.5*iqr_exp)

#Replacing outliers with NAN value.
for i in df["Experience"]:
  if i > upper_exp:
    df["Experience"].replace(i,np.nan, inplace = True)

#As the data is skewed, we will replace NAN value with Median value.
from sklearn.impute import SimpleImputer
knn_imputer_exp = SimpleImputer(strategy='median')
exp_reshape = np.array(df["Experience"]).reshape(-1, 1)
df["Experience"] = knn_imputer_exp.fit_transform(exp_reshape)
sns.histplot(df["Experience"], kde = True)
```

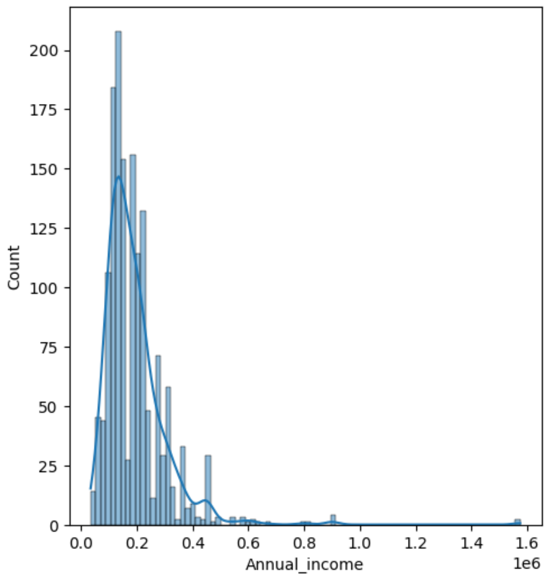
[21]:
```
<Axes: xlabel='Experience', ylabel='Count'>
```

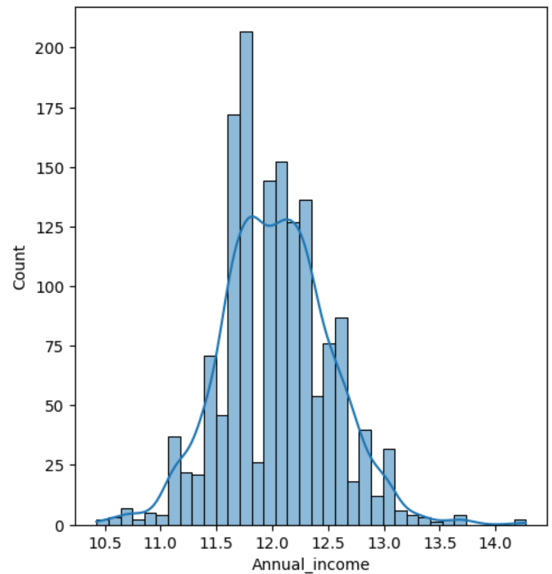


#####Column : Annual Income

[22]:
```
#Plot of Annual Income and Log(Annual Income).
plt.figure(figsize=(25,6))
plt.subplot(1, 4, 1)
sns.histplot(df["Annual_income"], kde = True)
print("Skewness of Annual Income : ", df["Annual_income"].skew())

plt.subplot(1, 4, 2)
sns.histplot(np.log(df["Annual_income"]), kde = True)
print("Skewness of Log of Annual Income : ", np.log(df["Annual_income"]).skew())
plt.show()
```

```
Skewness of Annual Income :  3.9245642452364167
Skewness of Log of Annual Income :  0.20152944215616836
```





[23]:
```
#Replacing the feature with transformed figure that is Log Transformation
df["Income_1"] = df['Annual_income'].copy()
df["Annual_income"] = np.log(df["Annual_income"])
```

[24]:
```
#There are some Null values after log transformation which affect the attribute␣
↪and ML Model.
print("Total Missing value after Log Transformation : ", df["Annual_income"].
↪isna().sum())

#Dropping the Null rows from the dataset.
df.drop(df[(df["Annual_income"].isna())].index, inplace=True)
```

```
Total Missing value after Log Transformation :  23
```

###Missing Values

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in a real-life scenarios. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed.

There are varioud techniques to handle missign values :- * Remove technique * Imputation technique

#####Imputation :- * Univariate Variable * Numerical Feature - Mean/ Median. * Categorical Feature - Mode or "Missing". * Multivariate Variable * KNN Imputation * Iterative (MICE) Imputation

[25]:
```
#There are Null values in GENDER, Age and Type of Occupation.
df.isnull().sum()
```

```
[25]: Ind_ID              0
      GENDER              7
      Car_Owner           0
      Propert_Owner       0
      CHILDREN            0
      Annual_income       0
      Type_Income         0
      EDUCATION           0
      Marital_status      0
      Housing_type        0
      Age                22
      Experience          0
      Mobile_phone        0
      Work_Phone          0
      Phone               0
      EMAIL_ID            0
      Type_Occupation   480
      Family_Members      0
      label               0
      exp1                0
      Income_1            0
      dtype: int64
```

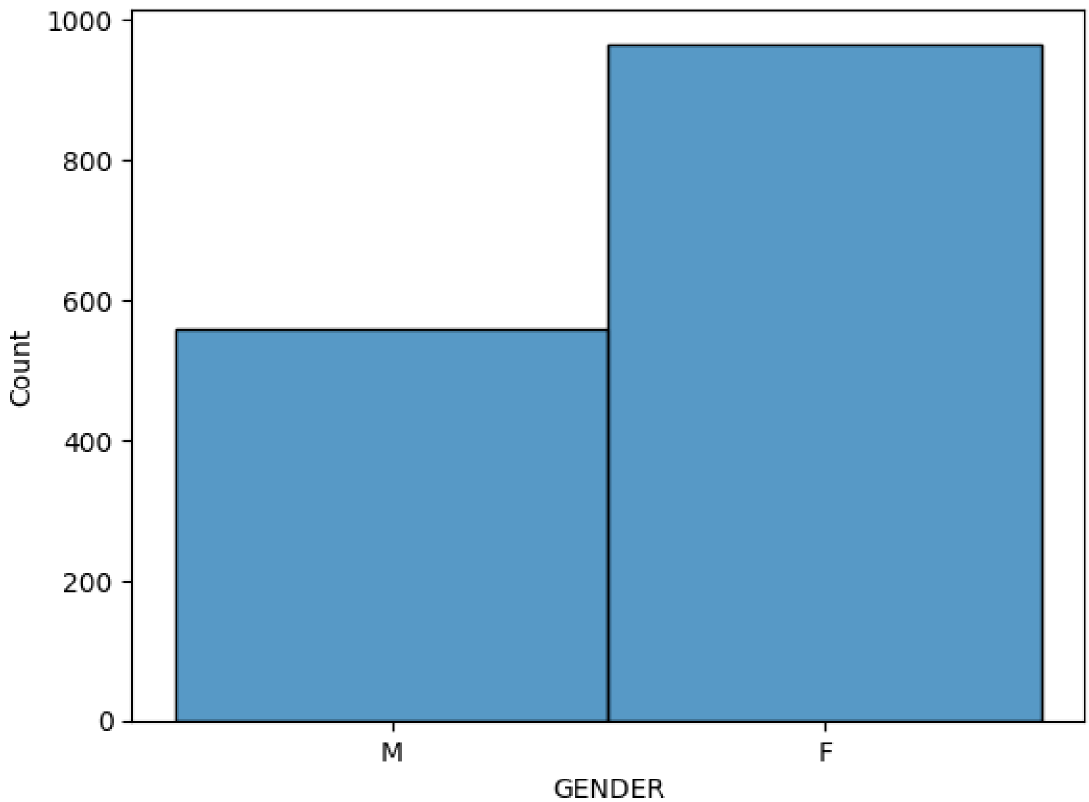
#####Column : Gender

```
[26]: #Counting the Female and Male counts from the dataset.
      df["GENDER"].value_counts()
```

```
[26]: F    959
      M    559
      Name: GENDER, dtype: int64
```

```
[27]: #Importing SimpleImputer from sklearn
      from sklearn.impute import SimpleImputer

      # It is Categorical(Nominal) - Gender
      # Defining "Most Frequent" to Null values as it has less than 5% of values.
      mode_imputer_gender = SimpleImputer(strategy='most_frequent')
      gender_reshape = np.array(df["GENDER"]).reshape(-1, 1)
      df["GENDER"] = mode_imputer_gender.fit_transform(gender_reshape)
      sns.histplot(df["GENDER"])
```

```
[27]: <Axes: xlabel='GENDER', ylabel='Count'>
```



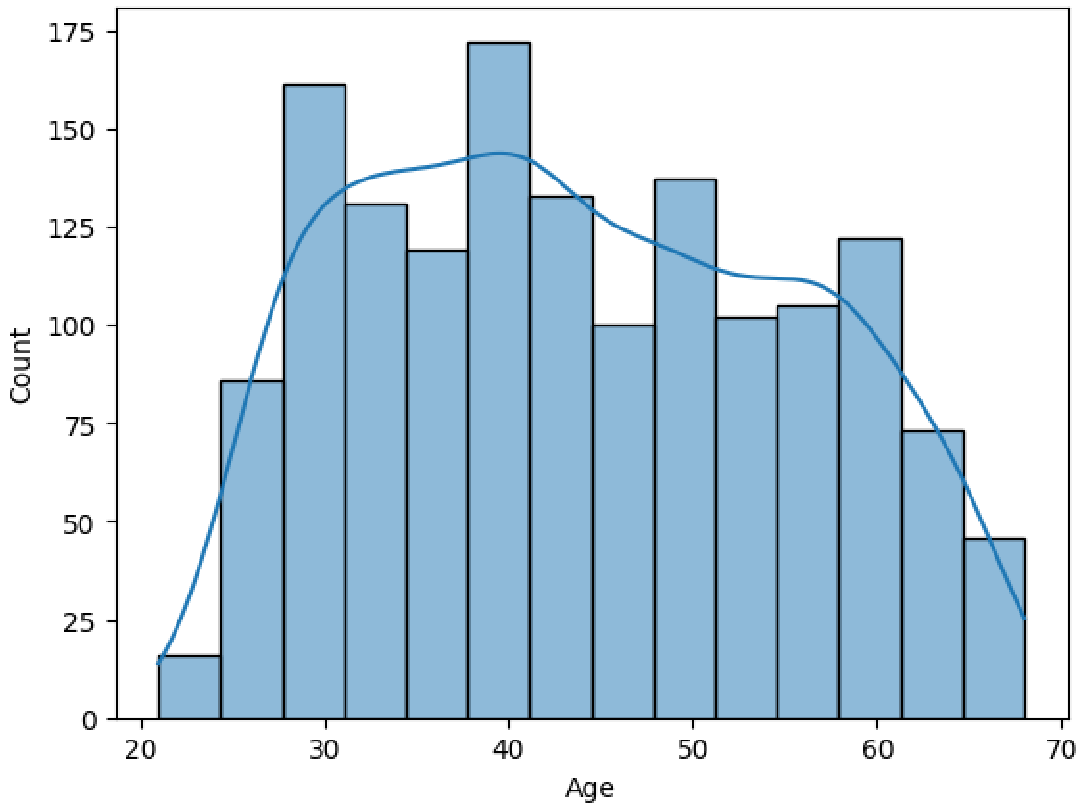
[28]:
```
df["GENDER"].value_counts()
#Increase of 7 values in "Female".
```

[28]:
```
F    966
M    559
Name: GENDER, dtype: int64
```

#####Column : Age

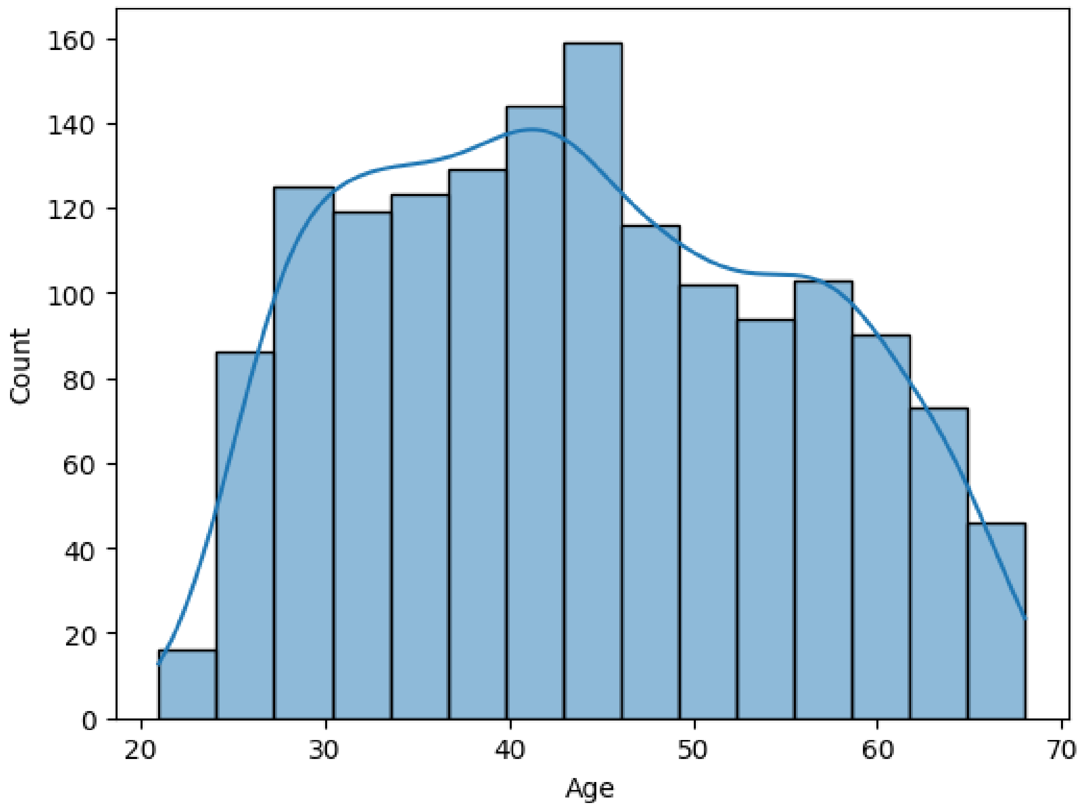
[29]:
```
sns.histplot(df["Age"], kde = True)
print("Null values in  Age feature is ",df["Age"].isnull().sum())
plt.show()
```

```
Null values in  Age feature is  22
```



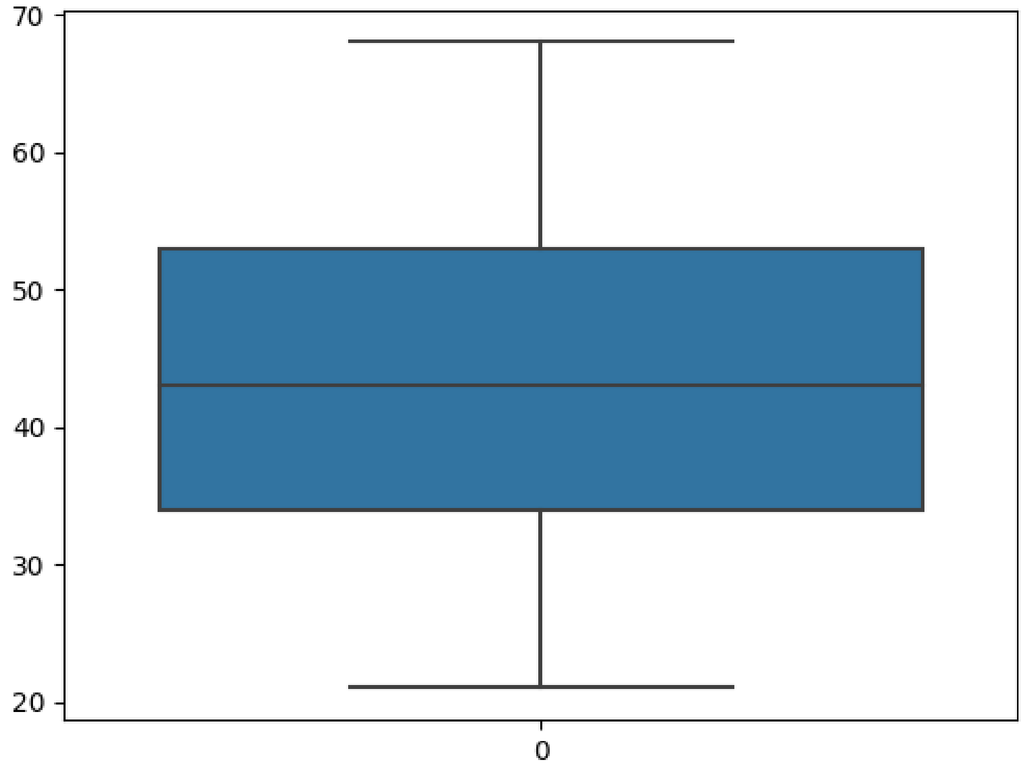
[30]:
```
#KNN Imputation : Identifies the neighboring points through a measure of␣
↪distance and the missing values can be estimated
#using completed values of neighboring observations.

from sklearn.impute import KNNImputer
knn_imputer_age = KNNImputer(n_neighbors=5)
income_reshape = np.array(df["Age"]).reshape(-1, 1)
df["Age"] = knn_imputer_age.fit_transform(income_reshape)
sns.histplot(df["Age"], kde = True)
```

[30]: <Axes: xlabel='Age', ylabel='Count'>



[31]:
```
#Outliers in Age using Boxplot.
sns.boxplot(df["Age"])
```

[31]: <Axes: >



#####Column : Type of Occupation

[32]:
```
#Extracting out the mean and null values from Type_Occupation.
print("Missing Value in Type of Occupation {} %".format(df['Type_Occupation'].
 ↪isna().mean()*100))
print("Total Missing Values : ",df['Type_Occupation'].isna().sum())
```

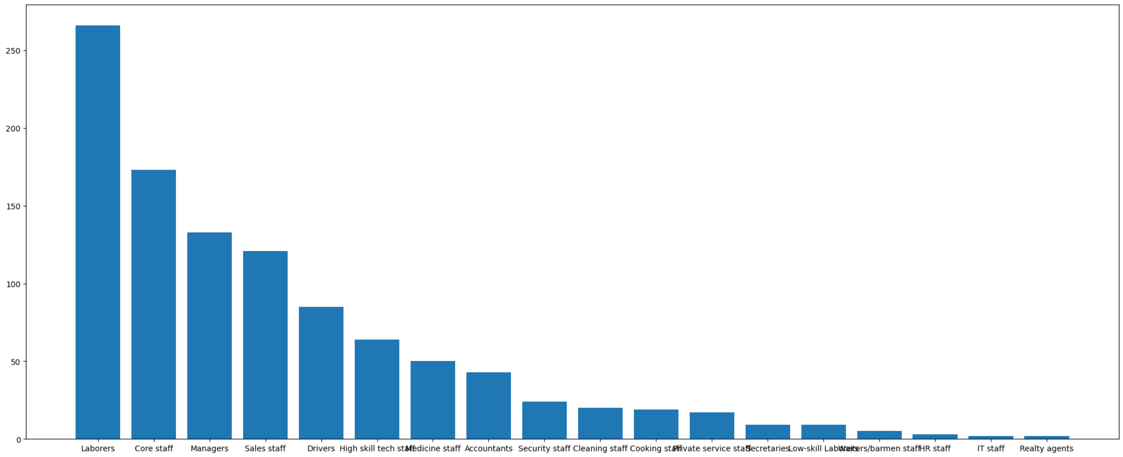
```
Missing Value in Type of Occupation 31.475409836065577 %
Total Missing Values :  480
```

Observation :- * Type_Occupation has 488 Missing values that is 31.5% of total value in the column.

[33]:
```
df["Type_Occupation"].unique()
```

[33]:
```
array([nan, 'Core staff', 'Cooking staff', 'Laborers', 'Sales staff',
       'Accountants', 'High skill tech staff', 'Managers',
       'Cleaning staff', 'Drivers', 'Low-skill Laborers', 'IT staff',
       'Waiters/barmen staff', 'Security staff', 'Medicine staff',
       'Private service staff', 'HR staff', 'Secretaries',
       'Realty agents'], dtype=object)
```

[34]:
```
plt.figure(figsize = (25,10))
plt.bar(df["Type_Occupation"].value_counts().index,df["Type_Occupation"].
 ↪value_counts().values)
plt.show()
```



[35]:
```
df.drop('Type_Occupation', axis = 1, inplace = True)
```

Obeseravtion :-

- This column has 31.5% of missing values so we are droping this.

###Outliers

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population.

There are various techniques to deal with Outliers :- * Trimming * Capping

These 2 things can be done under these concepts :- * Z-Score : Trimming and Capping at (mean ± 3* STD) * IQR Based Filtering * Percentile : Triiming and Capping at (99% and 1%) * Winsorization : Capping at (99% and 1%)

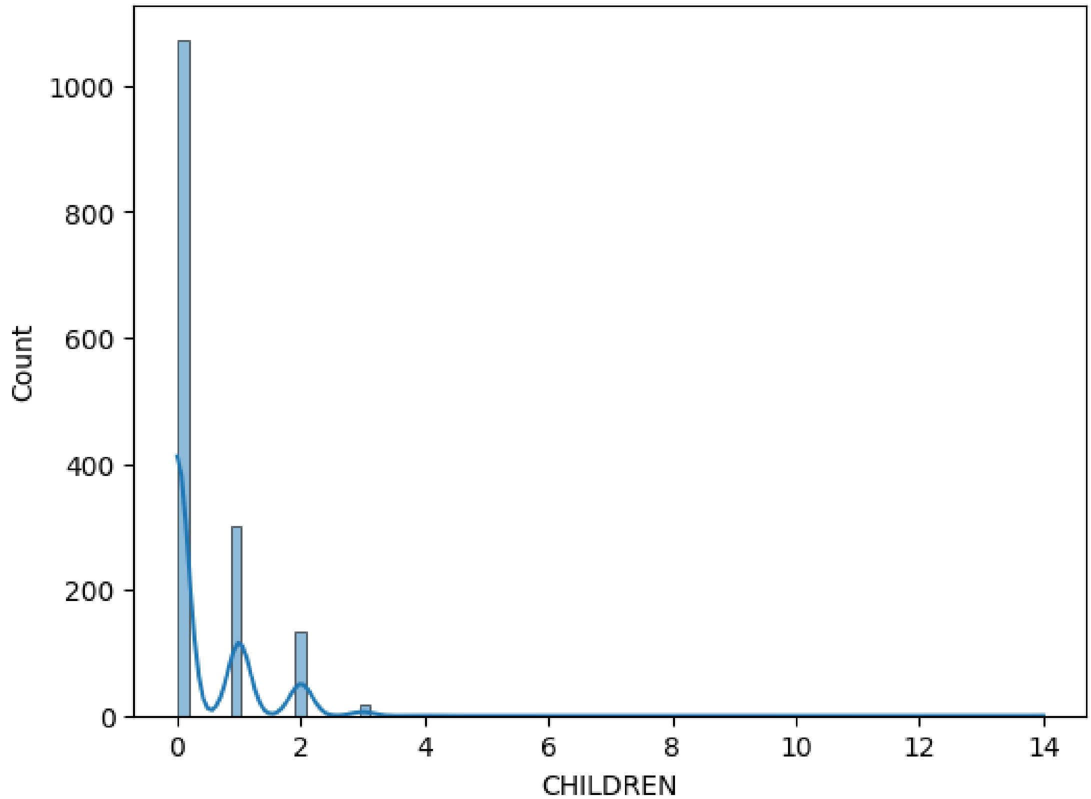
#####Column : Children

[36]:
```
df["CHILDREN"].value_counts()
```

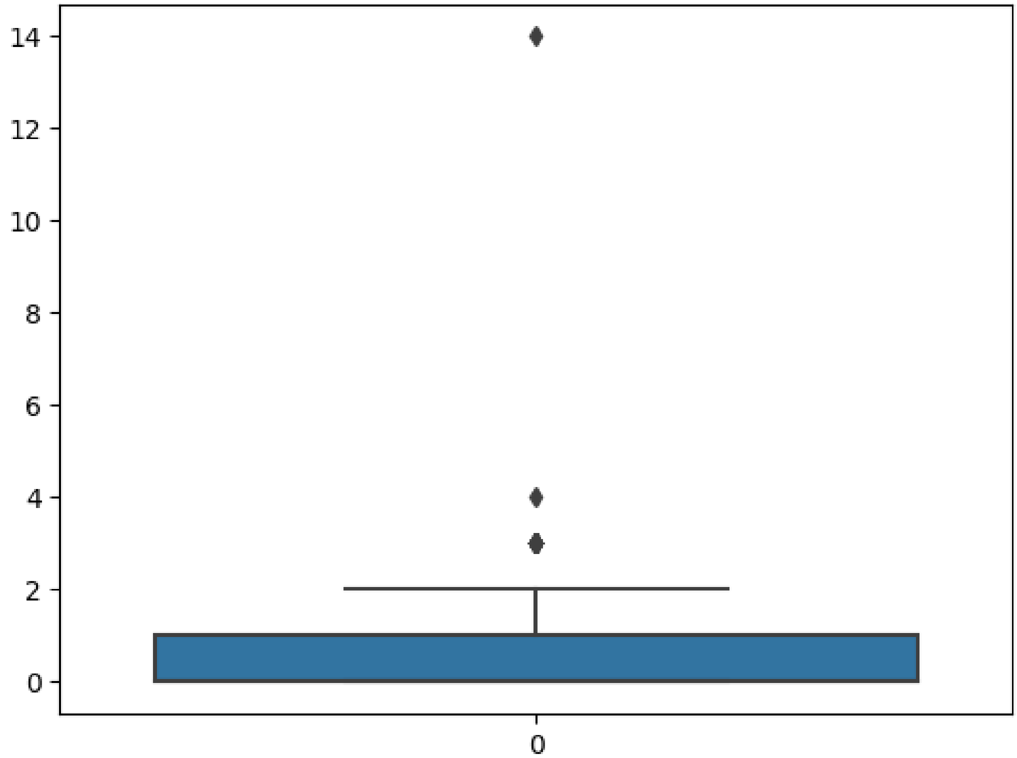
[36]:
```
0     1073
1      302
2      132
3       16
4        1
14       1
Name: CHILDREN, dtype: int64
```

[37]:
```
sns.histplot(df["CHILDREN"], kde = True)
```

[37]:
```
<Axes: xlabel='CHILDREN', ylabel='Count'>
```



[38]:
```
sns.boxplot(df["CHILDREN"])
plt.show()
```



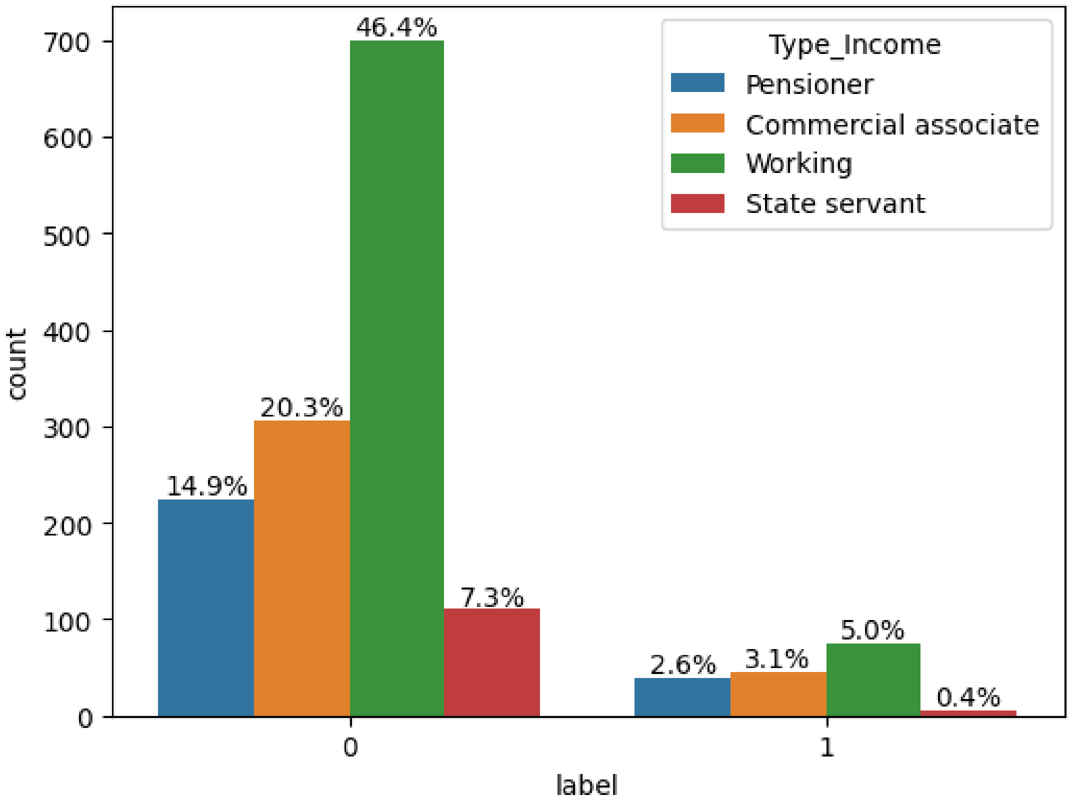
[39]:
```
q3_child = df["CHILDREN"].quantile(0.75)
q1_child = df["CHILDREN"].quantile(0.25)
iqr_child = q3_child - q1_child
upper_child = q3_child + (1.5*iqr_child)
lower_child = q1_child - (1.5*iqr_child)


outier_child = 0
for i in df["CHILDREN"]:
  if i > upper_child:
    outier_child+=1
print("Total rows removed :",outier_child)


df.drop(df[(df["CHILDREN"]>upper_child)].index, inplace=True)
df.drop(df[(df["CHILDREN"]<lower_child)].index, inplace=True)
```

```
Total rows removed : 18
```

#**EXPLORATORY DATA ANALYSIS**

#####BivariateAnalysis

[40]:
```
ax = sns.countplot(x = "label", hue = "Type_Income", data = df)

for i in ax.patches:
    percentage = '{:.1f}%'.format(100 * i.get_height()/len(df))
    x = i.get_x()+ i.get_width()/2
    y = i.get_height()+5
    ax.annotate(percentage, (x, y), ha='center')

plt.show()
```



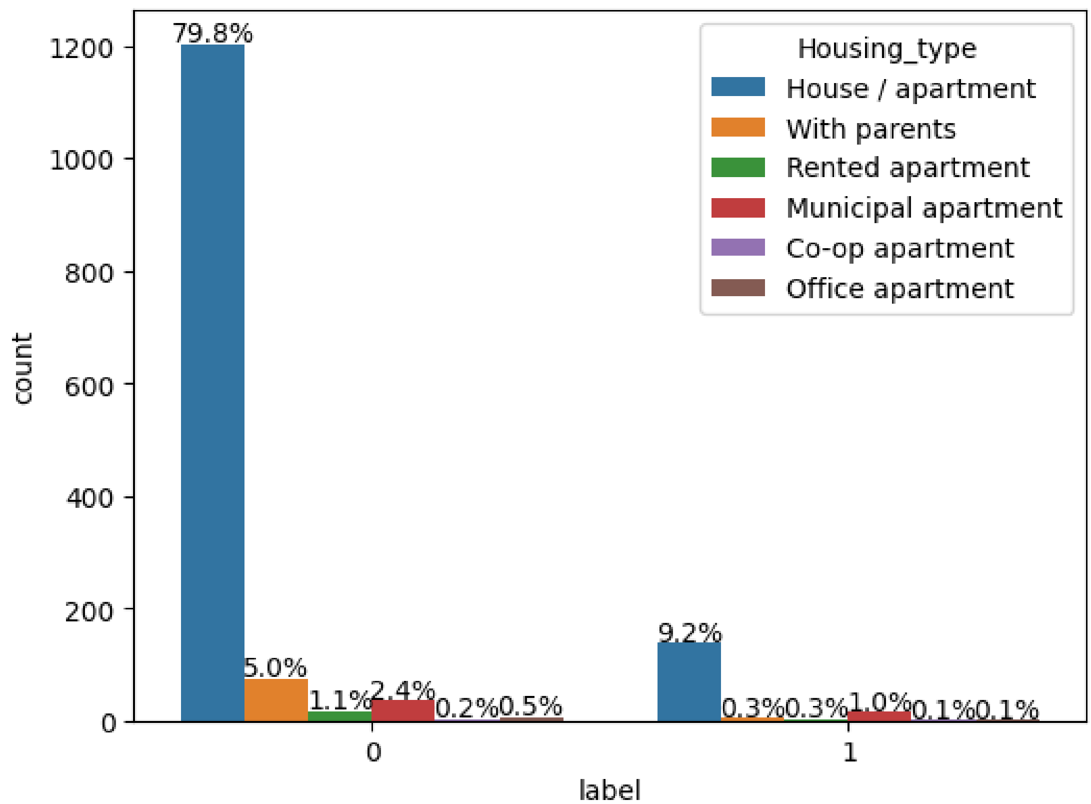
Obseravtion

- There are 4 types of Income
  - Pensioner
  - Commerical Associate
  - Working
  - State Servant
- Ther are more chances of getting Credit Card of "Working" as Compared to Other Income type.
- In the Same Category(Approval), There are **46.4%** more chances to get Credit Card to

working professinal.

- There is **88.6%** Credit Card Approved and **11.4%** Rejected on the basis of Income type.

[41]:
```
ax = sns.countplot(x = "label", hue = "Housing_type", data = df)
for i in ax.patches:
    percentage = '{:.1f}%'.format(100 * i.get_height()/len(df))
    x = i.get_x()+ i.get_width()/2
    y = i.get_height()+5
    ax.annotate(percentage, (x, y), ha='center')
plt.show()
```



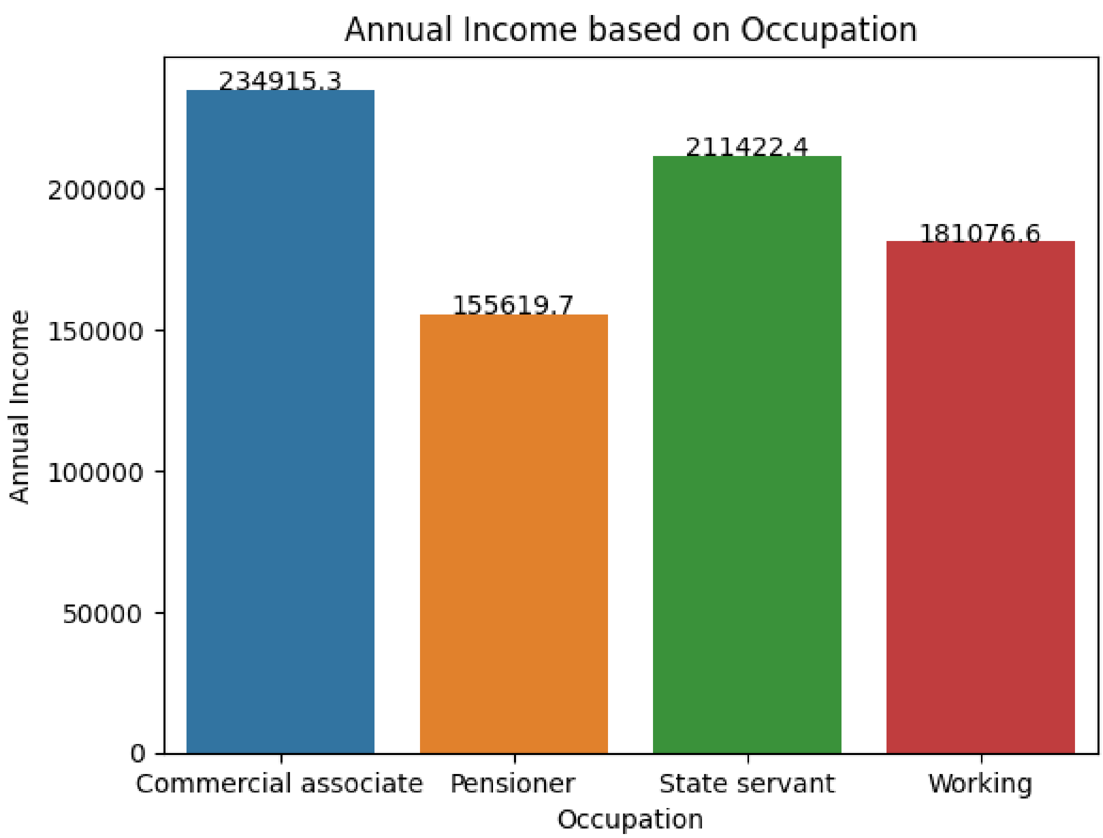
Obseravtion :-

- There are 6 types of Housing
  - House/Apartment
  - With Parents
  - Rented Apartment
  - Municipal Apartment
  - Co-op Apartment
  - Office Apartment
- Ther are more chances of getting Credit Card of "House Apartment" as Compared to Other

House type.

- In the Same Category(Approval), There are **79.8%** more chances to get Credit Card to House Apartment. Hence, we can say there is domination of Working in Housing Type. There is chances of removing this column from the dataset.
- There is **88.6%** Credit Card Approved and **11.4%** Rejected on the basis of Housing type.

[42]:
```
cat = df[["Type_Income", "Income_1"]].groupby(['Type_Income'], as_index=False).
  ↪mean()
ax = sns.barplot(x = "Type_Income", y = "Income_1", data = cat)

for i in ax.patches:
    percentage = '{:.1f}'.format(i.get_height())
    x = i.get_x()+ i.get_width()/2
    y = i.get_height()+5
    ax.annotate(percentage, (x, y), ha='center')

plt.title("Annual Income based on Occupation")
plt.xlabel("Occupation")
plt.ylabel("Annual Income")
plt.show()
```



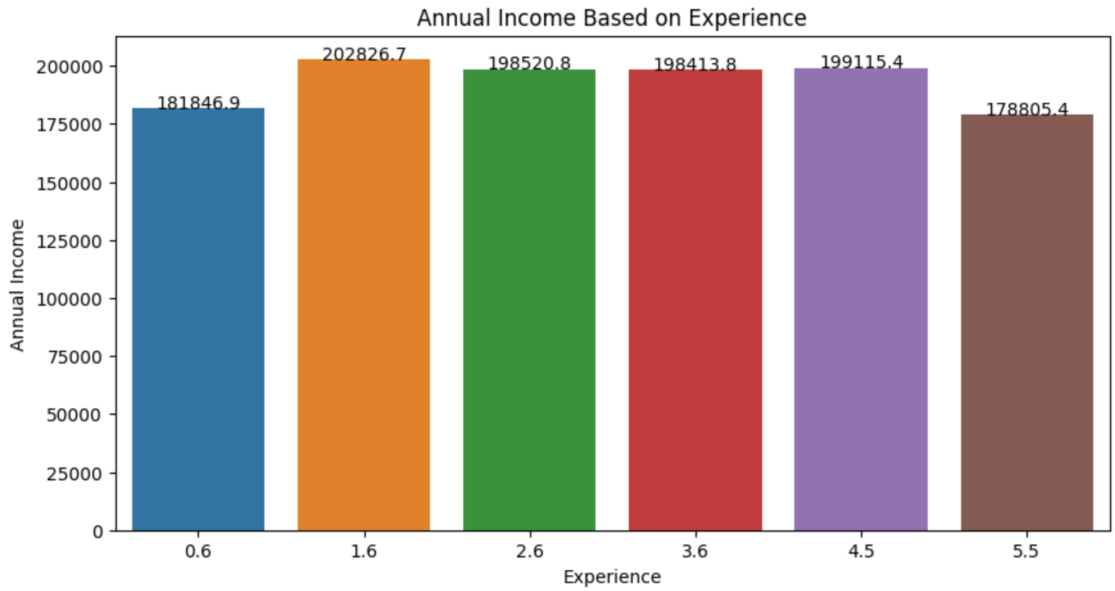
Obseravtion :-

- There are 4 types of Income
  - Pensioner
  - Commerical Associate
  - Working
  - State Servant
- The relationship between "Annual Income" and "Type of Income" show that Commerical associate and State servant are the highest among other and Occupation ranging from **1.56LPA** to **2.34LPA**.

[43]:

```
#Cutting the Experience by ranges from 0 to 22 and getting the average annual␣
↪Income based on the range of Experience.
exp1 = pd.cut(df["exp1"], bins = list(np.arange(0,7,1)))
print("Mean of Annual Income by the Age are :\n",df.groupby(exp1)["Income_1"].
↪mean(),"\n")

#Ploting the above code in Barchart for better understanding.
cat2 = round(df[['exp1',"Income_1"]].groupby(pd.cut(df["exp1"], bins = list(np.
↪arange(0,7,1))), as_index=False).mean(),1)
plt.figure(figsize =(10,5))
ax = sns.barplot(x = "exp1", y = "Income_1", data = cat2)

for i in ax.patches:
    percentage = '{:.1f}'.format(i.get_height())
    x = i.get_x()+ i.get_width()/2
    y = i.get_height()+5
    ax.annotate(percentage, (x, y), ha='center')

plt.title("Annual Income Based on Experience")
plt.xlabel("Experience")
plt.ylabel("Annual Income")
plt.show()
```

```
Mean of Annual Income by the Age are :
 exp1
(0, 1]    181846.934579
(1, 2]    202826.712329
(2, 3]    198520.833333
(3, 4]    198413.839286
(4, 5]    199115.384615
(5, 6]    178805.357143
Name: Income_1, dtype: float64
```



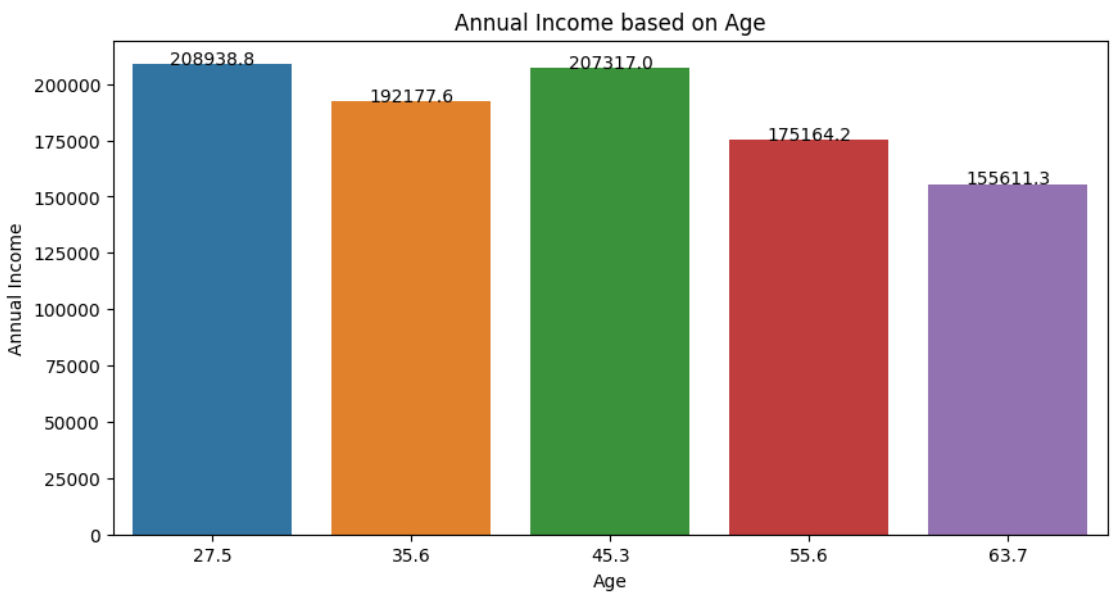
Obseravtion :-

- Highest Average Income observed from the chart ranging between 1 to 5 years of Experience that ranges from 1.99LPA to 2.02LPA.
- Lowest or Average Income observed is around 1.78LPA.

[44]:

```
#Cutting the bins from 20 to 70 in 10 each part to get the  average annual␣
↪income.
age = pd.cut(df["Age"], bins = [20,30,40,50,60,70])
print("Mean of Annual Income by the Age are :\n",df.groupby(age)["Income_1"].
↪mean(),"\n")

#Ploting the above code in Barchart for better understanding.
cat4 = round(df[['Age',"Income_1"]].groupby(pd.cut(df["Age"], bins =␣
↪[20,30,40,50,60,70]), as_index=False).mean(),1)
plt.figure(figsize =(10,5))
ax = sns.barplot(x = "Age", y = "Income_1", data = cat4)

for i in ax.patches:
    percentage = '{:.1f}'.format(i.get_height())
    x = i.get_x()+ i.get_width()/2
    y = i.get_height()+5
    ax.annotate(percentage, (x, y), ha='center')
plt.title("Annual Income based on Age")
plt.xlabel("Age")
plt.ylabel("Annual Income")
plt.show()
```

```
Mean of Annual Income by the Age are :
 Age
(20, 30]    208938.766520
(30, 40]    192177.573529
(40, 50]    207316.973684
(50, 60]    175164.199396
(60, 70]    155611.267606
Name: Income_1, dtype: float64
```

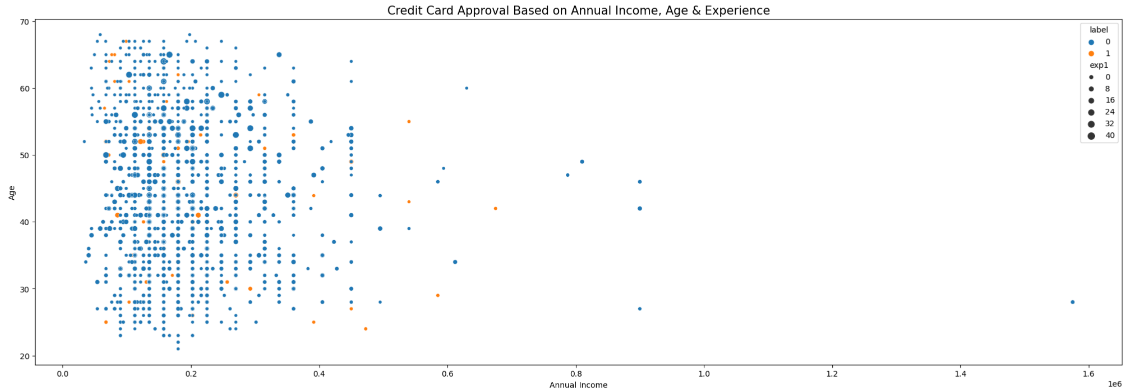


Obseravtion :-

- The relationship between "Annual Income" and "Age" ranging from **1.55LPA** to **2.09LPA**.

#####Multivariate Analysis

[45]:
```
plt.figure(figsize = (25,8))

#Plotting "Age" and "Annual Income" plot on the basis of label and Experience.
sns.scatterplot(x = df['Income_1'],y = df["Age"], hue = df["label"], size =␣
 ↪df["exp1"])
plt.title("Credit Card Approval Based on Annual Income, Age & Experience",␣
 ↪fontsize = 15)
plt.xlabel("Annual Income")
plt.ylabel("Age")
plt.show()
```



Obseravtion :-

- Experience is divided into 0-8, 8-16, 16-24, 24-32, 32-40 years with the label encoded 0(Approved) or 1(Rejected).
- There is high probabity to get the Credit Card Approval in the Age group 22 to 60 years.
- Annual Income ranging from 1LPA to (2-2.5)LPA also shows high chances to get the Credit Card Approval.

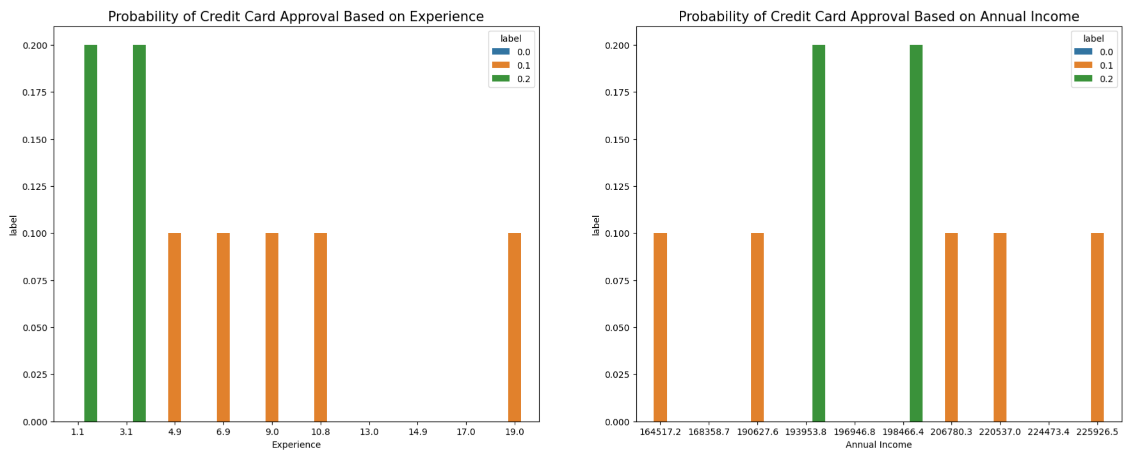
Conclusion :-

- Those people who have **4 to 16 years of Experience** and **Age 30-55 years** and **Annual Income ranging from 1.5 to 2 LPA** has High probabity to get the Credit Card Approval.
- From this, we can say **"Age"**, **"Experience"** and **"Annual Income"** are Crucial Features for the Feature Selection.

[46]:
```
exp1 = pd.cut(df["exp1"], bins = list(np.arange(0,22,2)))
print("Mean of Annual Income by the Age are :\n",df.groupby(exp1)["label"].mean(),"\n")

cat3 = round(df[['exp1',"Income_1", "label"]].groupby(pd.cut(df["exp1"], bins = list(np.arange(0,22,2))), as_index=False).mean(),1)
plt.figure(figsize =(45,8))

plt.subplot(1, 4, 1)
sns.barplot(x = "exp1", y = "label", data = cat3, hue = "label")
plt.title("Probability of Credit Card Approval Based on Experience", fontsize = 15)
plt.xlabel("Experience")
plt.ylabel("label")

plt.subplot(1, 4, 2)
sns.barplot(x = "Income_1", y = "label", data = cat3, hue = "label")
```

```
plt.title("Probability of Credit Card Approval Based on Annual Income",␣
↪fontsize = 15)
plt.xlabel("Annual Income")
plt.ylabel("label")
plt.show()
```

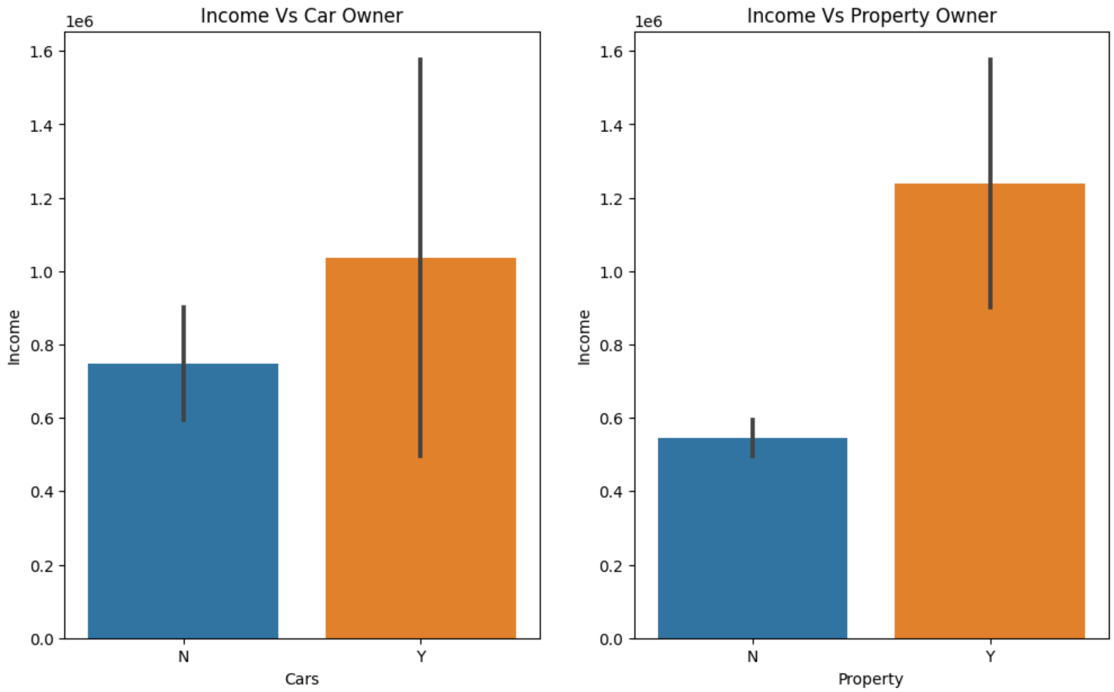
```
Mean of Annual Income by the Age are :
 exp1
(0, 2]      0.162055
(2, 4]      0.159091
(4, 6]      0.089552
(6, 8]      0.069620
(8, 10]     0.094488
(10, 12]    0.088235
(12, 14]    0.042553
(14, 16]    0.000000
(16, 18]    0.000000
(18, 20]    0.125000
Name: label, dtype: float64
```



OBSERVATIONS :- * People have 0 to 4 Year of Experience have 20% chances to get the Credit Card. * Salary between 1.9 to 2.1 Lakhs Lakhs has 20% chances to get Credit Card.

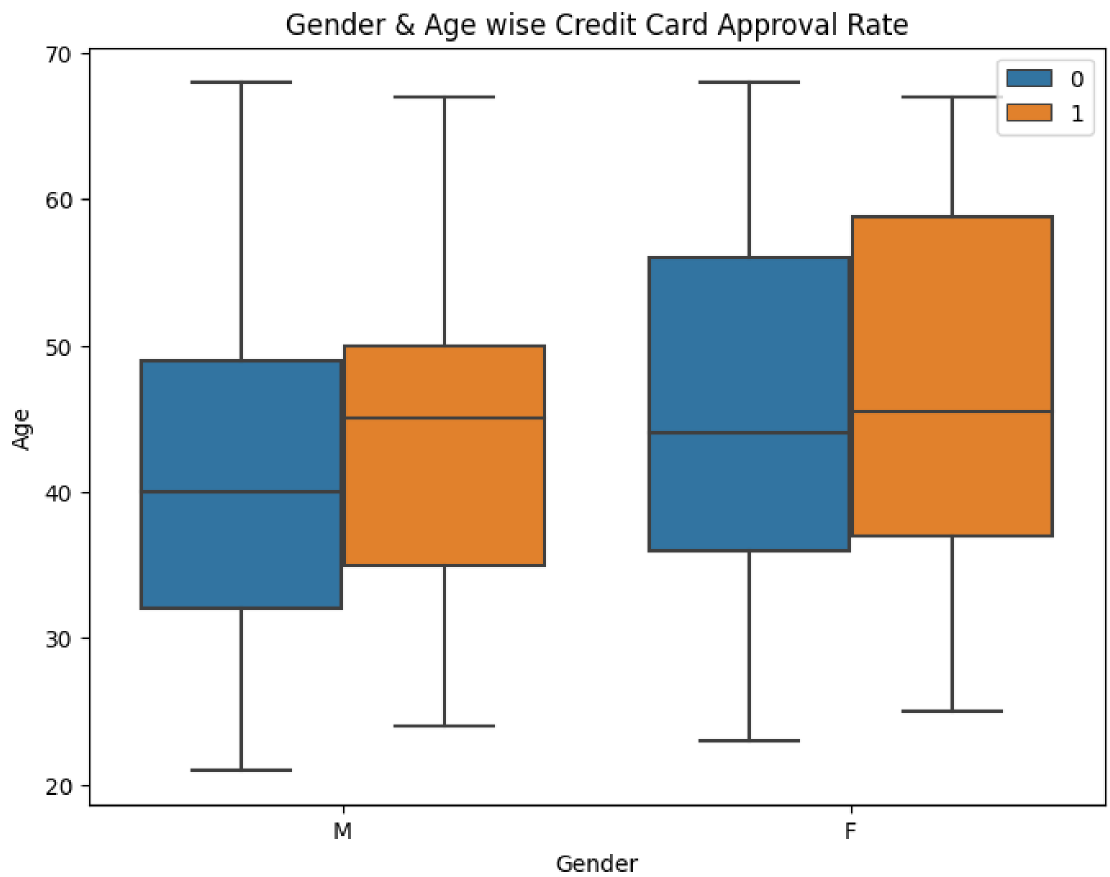
[47]:
```
cat1 = df[["Car_Owner","Propert_Owner", "Income_1"]].
↪groupby(['Car_Owner','Propert_Owner'], as_index=False).agg({'Income_1':
↪'count','Income_1':'max'})

plt.figure(figsize = (25,7))
plt.subplot(1, 4, 1)
sns.barplot(x= "Car_Owner", y = "Income_1", data = cat1)
plt.title("Income Vs Car Owner")
```

```
plt.xlabel("Cars")
plt.ylabel("Income")

plt.subplot(1, 4, 2)
sns.barplot(x = "Propert_Owner", y = "Income_1", data = cat1)
plt.title("Income Vs Property Owner")
plt.xlabel("Property")
plt.ylabel("Income")
plt.show()
```



OBSERVATION :- * Income between 0.8LPA to 1LPA have Car. * Income between 0.6LPA to 1.2LPA have Property.

[48]:
```
plt.figure(figsize = (8,6))
sns.boxplot(x = "GENDER", y = "Age", data = df, hue = "label")
plt.title("Gender & Age wise Credit Card Approval Rate")
plt.xlabel("Gender")
plt.ylabel("Age")
plt.legend(loc = "upper right")
plt.show()

print(df.groupby(["label", "GENDER"])[["GENDER"]].count())
```




```
             GENDER
label GENDER
0     F         861
      M         480
1     F          94
      M          72
```

OBSERVATIONS :- * Male has 35.84% chances and Female with **64.2%** chances to get the Credit Card Approval. * From all dataset, Female has **57.28%** chances and Male has **31.93%** chances to get Credit Card Approval.

Conclusions :-

- From the observation, we can say that "Female" can claim Credit Card on the ratio of **0.55 times more** than "Male".

#**FEATURE SCALING**

- Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization.

####Feature Selection

- Feature Selection is the method of reducing the input variable to your model by using only relevant data and getting rid of noise in data.

[49]:
```
#"Corr" use to get the relationship between all numerical variables.
## It ranges between -1 to 1.
## 0 to 1 indicates : Directly correlated with each other.
## -1 to 0 indicates : Inversly correlated with each other.
df.corr()
```

[49]:
```
                 Ind_ID  CHILDREN  Annual_income       Age  Experience  \
Ind_ID         1.000000  0.034111       0.010696 -0.021196    0.024035
CHILDREN       0.034111  1.000000       0.077182 -0.324860    0.158094
Annual_income  0.010696  0.077182       1.000000 -0.129941    0.154572
Age           -0.021196 -0.324860      -0.129941  1.000000   -0.252684
Experience     0.024035  0.158094       0.154572 -0.252684    1.000000
Mobile_phone        NaN       NaN            NaN       NaN         NaN
Work_Phone     0.086924  0.038923      -0.069047 -0.170643    0.178011
Phone          0.010911 -0.002851      -0.014129  0.029284    0.016499
EMAIL_ID      -0.040368  0.042872       0.128075 -0.165706   -0.007579
Family_Members 0.013259  0.861058       0.051906 -0.289038    0.183507
label          0.019160 -0.017703       0.020485  0.037078   -0.087980
exp1           0.003828  0.072266       0.087537 -0.020329    0.884023
Income_1       0.033354  0.097653       0.897130 -0.111962    0.108323

                Mobile_phone  Work_Phone     Phone  EMAIL_ID  Family_Members  \
Ind_ID                   NaN    0.086924  0.010911 -0.040368        0.013259
CHILDREN                 NaN    0.038923 -0.002851  0.042872        0.861058
Annual_income            NaN   -0.069047 -0.014129  0.128075        0.051906
Age                      NaN   -0.170643  0.029284 -0.165706       -0.289038
Experience               NaN    0.178011  0.016499 -0.007579        0.183507
Mobile_phone             NaN         NaN       NaN       NaN             NaN
Work_Phone               NaN    1.000000  0.348958 -0.006183        0.080230
Phone                    NaN    0.348958  1.000000  0.017704        0.005318
EMAIL_ID                 NaN   -0.006183  0.017704  1.000000        0.049817
Family_Members           NaN    0.080230  0.005318  0.049817        1.000000
label                    NaN   -0.011823 -0.009692  0.010688       -0.026954
exp1                     NaN    0.127173  0.037139 -0.038656        0.089362
Income_1                 NaN   -0.070595 -0.004622  0.121593        0.059665

                  label      exp1  Income_1
Ind_ID         0.019160  0.003828  0.033354
CHILDREN      -0.017703  0.072266  0.097653
Annual_income  0.020485  0.087537  0.897130
Age            0.037078 -0.020329 -0.111962
Experience    -0.087980  0.884023  0.108323
Mobile_phone        NaN       NaN       NaN
Work_Phone    -0.011823  0.127173 -0.070595
```

```
Phone          -0.009692  0.037139 -0.004622
EMAIL_ID        0.010688 -0.038656  0.121593
Family_Members -0.026954  0.089362  0.059665
label           1.000000 -0.095278  0.027429
exp1           -0.095278  1.000000  0.051116
Income_1        0.027429  0.051116  1.000000
```

[50]:
```
#Droping soem columns which as least chances or no chances to increase accuracy␣
↪of ML model.
df.drop(["Ind_ID", "Marital_status", "Mobile_phone", "Work_Phone", "Phone",␣
↪"EMAIL_ID", "exp1", "Income_1"], axis=1, inplace =True)
```

[51]:
```
#Dividing into Independent and Dependent Variable.
X = df.drop(["label"], axis=1)
y = df["label"]
```

[52]:
```
#To the dummy columns from all unique values in each columns.
X1 = pd.get_dummies(X)
```

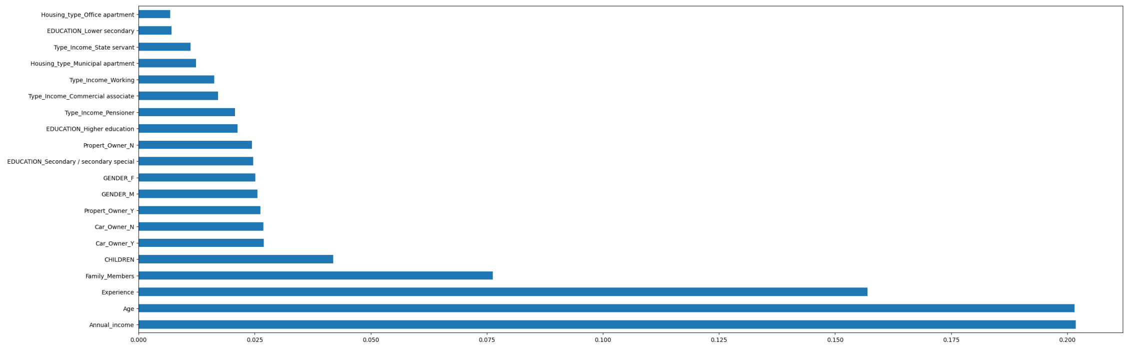
[53]:
```
#Importing ExtratreeRegressor from sklearn
from sklearn.ensemble import ExtraTreesRegressor
model = ExtraTreesRegressor()
model.fit(X1,y)
```

[53]:
```
ExtraTreesRegressor()
```

[54]:
```
#Getting all the feature importances with target variable.
print(model.feature_importances_)
```

```
[4.19259242e-02 2.01873158e-01 2.01535673e-01 1.56978078e-01
 7.62905683e-02 2.51291236e-02 2.56393077e-02 2.68484327e-02
 2.69484594e-02 2.44312057e-02 2.62163404e-02 1.71756091e-02
 2.08171297e-02 1.11973686e-02 1.63035476e-02 3.53029231e-05
 2.13273035e-02 6.19384818e-03 7.07636713e-03 2.47431220e-02
 6.29584748e-03 6.21278523e-03 1.23889635e-02 6.85759878e-03
 5.39586175e-03 4.16307316e-03]
```

[55]:
```
#plotting all the important columns which are related to target variable.
feat_important = pd.Series(model.feature_importances_, index = X1.columns)
plt.figure(figsize= (30,10))
#Plotting TOP-20 Columns from the Dummy variable "X1".
feat_important.nlargest(20).plot(kind="barh")
plt.show()
```



OBSERVATION * TOP-20 column are correlated to target variable. * Top 5 are list below:- * Age * Annual Income * Experience * Family Members * Children

####Splitting Data

[56]:
```python
#Splitting into training and test dataset with 75% and 25% on random state=0.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=0)
```

[57]:
```python
#Getting the shape of training and test dataset.
print("Train : ",X_train.shape, y_train.shape)
print("Test : ",X_test.shape, y_test.shape)
```

```
Train :  (1130, 11) (1130,)
Test :  (377, 11) (377,)
```

[58]:
```python
X_train.head()
```

[58]:
```
     GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
1433      F         N             N         0      12.100712
735       F         N             N         2      11.779129
258       F         Y             N         0      11.707670
690       M         Y             Y         0      11.407565
1231      F         N             N         0      11.630709

                Type_Income                      EDUCATION       Housing_type  \
1433  Commercial associate  Secondary / secondary special  House / apartment
735                Working  Secondary / secondary special  House / apartment
258              Pensioner  Secondary / secondary special  House / apartment
690              Pensioner  Secondary / secondary special  House / apartment
1231               Working  Secondary / secondary special  House / apartment

       Age  Experience  Family_Members
1433  31.0    2.549510               1
```

```
735   42.0   3.082207          4
258   64.0   0.000000          1
690   50.0   0.000000          2
1231  34.0   0.707107          1
```

####Categorical Encoding

- Encoding categorical data is a process of converting categorical data into integer format so that the data with converted categorical values can be provided to the different models.

[59]:
```
# Importing ColumnTransformer, OneHotEncoder & OrdinalEncoder.
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder


transformer = ColumnTransformer(transformers = [
        ("cat1", OneHotEncoder(sparse = False, drop = "first"), [0,1,2,5,7]),
        ("cat2", OrdinalEncoder(categories=[["Incomplete higher", "Lower␣
 ↪secondary", "Secondary / secondary special",
                                            "Academic degree", "Higher␣
 ↪education"]]), [6])
                ], remainder= "passthrough")



#Using fit_transform to the train data.
X_train = transformer.fit_transform(X_train)

#"transform" only to avoid data leakage.
X_test = transformer.transform(X_test)
```

####SMOTE Technique

SMOTE (Synthetic Minority Over Sampling) * SMOTE is a machine learning technique that solves problems that occur when using an imbalanced data set. Imbalanced data sets often occur in practice, and it is crucial to master the tools needed to work with this type of data. * SMOTE is a solution when you have imbalanced data.

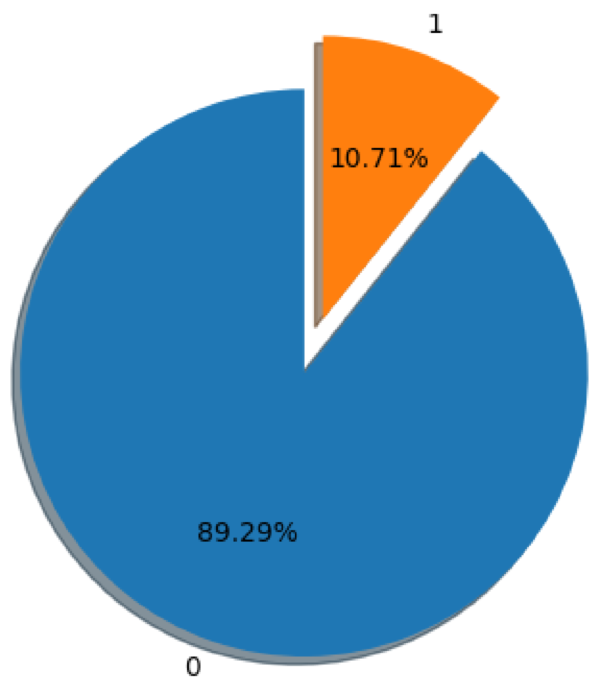
- Resampling can help to improve model performance in cases of imbalanced data sets. It creates new samples by selecting data points randomly from the original dataset, and these new samples can be used to estimate the population characteristics of the data or to test the performance of a machine learning model.
  - There are several techniques of data resampling, which can be broadly classified into two categories:
    1. Undersampling: This involves reducing the number of samples in the majority class to balance the class distribution with the minority class. Drawback of this technique

is that it can result in loss of useful information.

2. Oversampling: This involves increasing the number of samples in the minority class to balance the class distribution with the majority class. This technique may lead to overfitting.

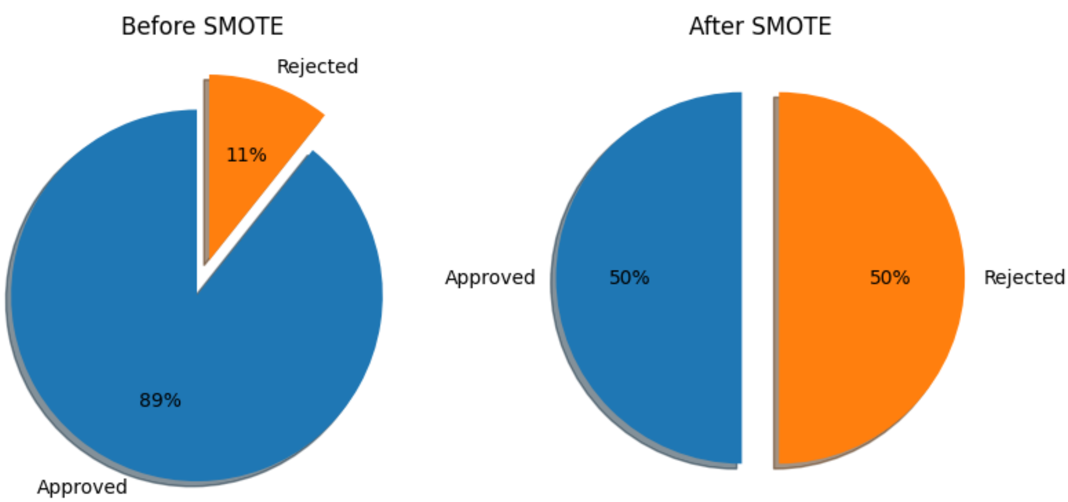
- SMOTE works by creating synthetic samples of the minority class, based on those that already exist. It works randomly picking a point from the minority class and computing the K-Nearest Neighbors for this point. The synthetic points are added between the chosen point and its neighbors.

[60]:
```
#Using Pie Chart, Plotting the Label feature without SMOTE technique.
plt.pie(y_train.value_counts(), labels = y_train.unique(), startangle = 90,␣
↪shadow = True, explode=(0.1, 0.1), autopct = '%1.2f%%')
plt.show()
```



[61]:
```
#Importing the SMOTE from the imblearn.
from imblearn.over_sampling import SMOTE

#Defining the smote for SMOTE with neighbour 5 and random state 130.
smote = SMOTE(random_state = 130, k_neighbors = 5)

# Fitting and resampling training data values
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
```

[62]:
```
#Plotting the Before SMOTE and After SMOTE pie chart.
plt.figure(figsize=(20,15))
plt.subplot(1,4,1)
plt.pie(y_train.value_counts(), labels = ["Approved", "Rejected"], startangle =␣
↪90, shadow = True, explode=(0.1, 0.1), autopct = '%1.0f%%')
plt.title("Before SMOTE")

plt.subplot(1,4,2)
plt.pie(y_train_sm.value_counts(), labels = ["Approved", "Rejected"],␣
↪startangle = 90, shadow = True, explode=(0.1, 0.1), autopct = '%1.0f%%')
plt.title("After SMOTE")
plt.show()
```



OBSERVATION * Before Approval rate was 89% and Rejection rate was 11%. * After SMOTE technique, Both are Equal.

####Standardization

- This technique is to re-scale features value with the distribution value between 0 and 1 is useful for the optimization algorithms, such as gradient descent, that are used within machine learning algorithms that weight inputs.
  - Normalization - Here we use Min-Max Scaler formula in scaling.
  - Standardization - Here we use Z-Score formula in scaling.

[63]:
```
#Importing Standardization from sklearn
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
#Using fit_transform to the train data.
X_train = scaler.fit_transform(X_train)
X_train_sm = scaler.fit_transform(X_train_sm)

#"transform" only to avoid data leakage.
X_test = scaler.transform(X_test)
```

#**MODEL SELECTION**

##**LOGISTIC REGRESSION**

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

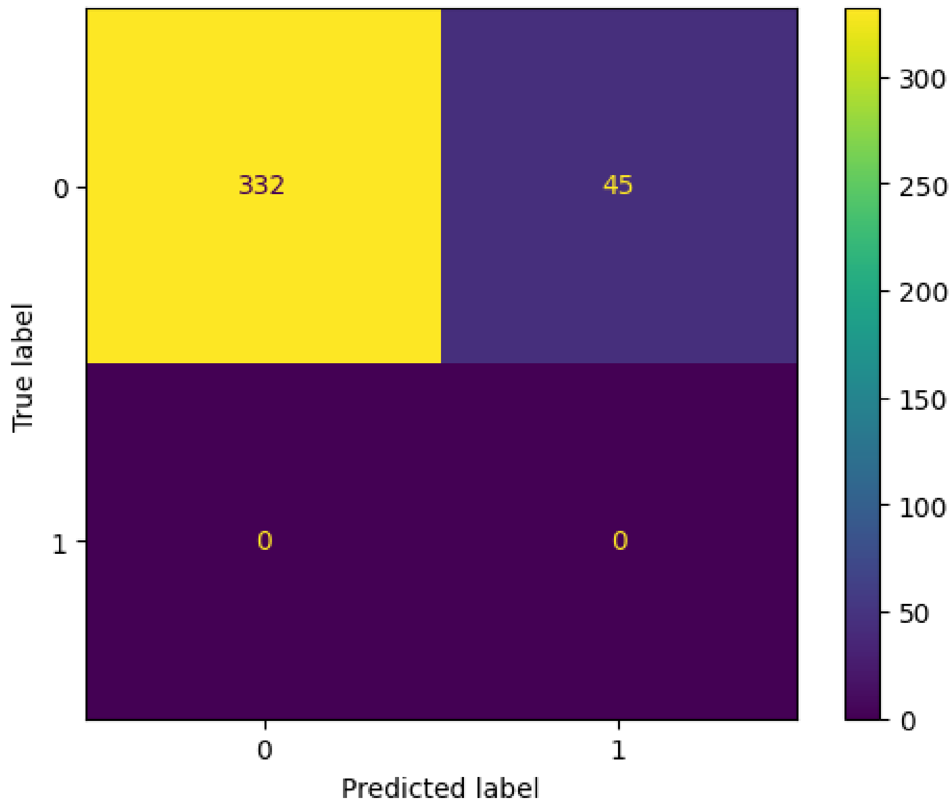
#####Baseline Logistic Regression

[64]:
```
#Importing Logistic Regression from sklearn.
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()

#Fitting the training independent and dependent variable to get trained.
log_reg.fit(X_train,y_train)

#Predicting the result foe the test dataset "X_test".
y_pred_log = log_reg.predict(X_test)
```

[65]:
```
#Importing accuracy_score, confusion_matrix, classification_report from sklearn.
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report, ConfusionMatrixDisplay

#Calculating the accuracy for the predicted and actual dataset.
Log_reg_untune = accuracy_score(y_pred_log, y_test)*100
print("Accuracy of Logistic Regression Model : {}%".
↪format(round(accuracy_score(y_pred_log, y_test)*100)))
```

```
Accuracy of Logistic Regression Model : 88%
```

[66]:
```
#Calculating for the confusion matrix.
cm_log = confusion_matrix(y_pred_log, y_test, labels= log_reg.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm_log, display_labels=log_reg.
↪classes_)
disp.plot()
plt.show()
```



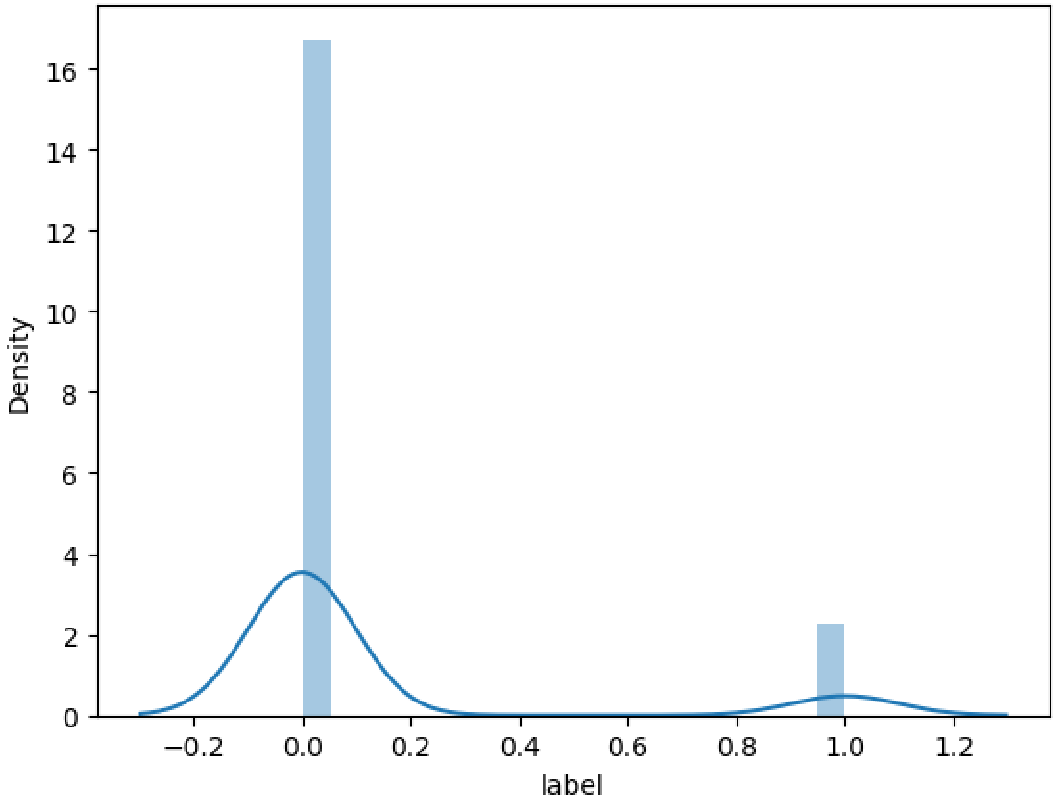
OBSERVATIONS * Model is predicting the following:- * TP - “332” people get approval of credit card. * TN - “0” people not get approval of credit card. * FP - “45” people not get approval of credit card but model predicted they will get. * FN - “0” people get approval of credit card but model predicted they will not get.

[67]:
```
#Getting the Classification report for precision, recall & f1-score value.
print("Classification Report of Logistic Regression Model :␣
↪\n",classification_report(y_pred_log, y_test))
```

```
Classification Report of Logistic Regression Model :
               precision    recall  f1-score   support

           0       1.00      0.88      0.94       377
           1       0.00      0.00      0.00         0

    accuracy                           0.88       377
   macro avg       0.50      0.44      0.47       377
weighted avg       1.00      0.88      0.94       377
```

[68]:
```python
#Plotting the test - predicted graph.
sns.distplot(y_test - y_pred_log)
```

[68]: <Axes: xlabel='label', ylabel='Density'>



OBSERVATION * It follows Normal Distribution Curve. * There are some noise(error) at 1(value).

#####SMOTE Logistic Regresion

[69]:
```python
log_reg_smote = LogisticRegression()
log_reg_smote.fit(X_train_sm, y_train_sm)

#Predicting the result foe the test dataset "X_test".
log_reg_smotee = log_reg_smote.predict(X_test)

print("Accuracy of Logistic Regression Model : {}%".
 ↪format(round(accuracy_score(log_reg_smotee, y_test)*100)))
lg_smote_score = accuracy_score(log_reg_smotee, y_test)*100
#Calculating for the confusion matrix.
confusion_matrix(log_reg_smotee, y_test)
```

```
Accuracy of Logistic Regression Model : 59%
```

[69]:
```
array([[196,  20],
       [136,  25]])
```

OBSERVATION :- * SMOTE doesn't help in model improvement as Type-1 and Type-2 error are high.

#####Hyperparameter Tuning for Logistic Regression

- Hyperparameter tuning consists of finding a set of optimal hyperparameter values for a learning algorithm while applying this optimized algorithm to any data set. That combination of hyperparameters maximizes the model's performance, minimizing a predefined loss function to produce better results with fewer errors.

[70]:
```python
#Importing GridSearchCV from sklearn
from sklearn.model_selection import GridSearchCV
parameter = {"penalty":["l1", "l2", "elasticnet"],
             "C" : [1,2,3,4,5,6,10,20,40,50],
             "max_iter" : [100,200,300,400,500]}
Classfier_Reg = GridSearchCV(log_reg,param_grid=parameter, scoring="accuracy",␣
↪cv=10)
```

[71]:
```python
#Fitting the training independent and dependent variable to get trained.
Classfier_Reg.fit(X_train, y_train)
```

[71]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [1, 2, 3, 4, 5, 6, 10, 20, 40, 50],
                         'max_iter': [100, 200, 300, 400, 500],
                         'penalty': ['l1', 'l2', 'elasticnet']},
             scoring='accuracy')
```

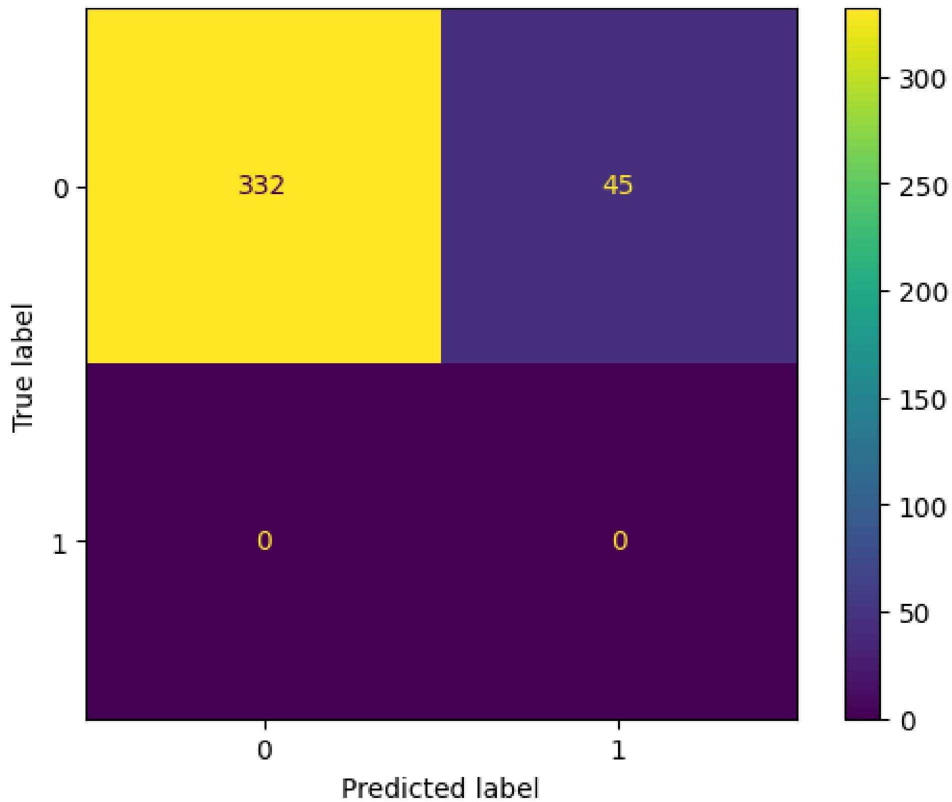
[72]:
```python
#Getting the best parameter from gridsearchcv
print(Classfier_Reg.best_params_)

#getting the best score for training the model.
print(Classfier_Reg.best_score_)

#Predicting the result foe the test dataset "X_test".
y_pred__log_tune = Classfier_Reg.predict(X_test)
```

```
{'C': 1, 'max_iter': 100, 'penalty': 'l2'}
0.8929203539823009
```

[73]:
```python
#Calculating the accuracy for the predicted and actual dataset.
Log_reg_tune = accuracy_score(y_pred__log_tune, y_test)*100
round(accuracy_score(y_pred__log_tune, y_test)*100)
```

[73]:
```
88
```

[74]:
```
#Calculating for the confusion matrix.
cm_log_tune = confusion_matrix(y_pred__log_tune, y_test, labels= Classfier_Reg.
 ↪classes_)
disp1 = ConfusionMatrixDisplay(confusion_matrix=cm_log_tune,␣
 ↪display_labels=Classfier_Reg.classes_)
disp1.plot()
plt.show()
```



[75]:
```
#Getting the Classification report for precision, recall & f1-score value.
print("Classification Report of Logistic Regression Model :␣
 ↪\n",classification_report(y_pred__log_tune, y_test))
```

```
Classification Report of Logistic Regression Model :
               precision    recall  f1-score   support

           0       1.00      0.88      0.94       377
           1       0.00      0.00      0.00         0

    accuracy                           0.88       377
   macro avg       0.50      0.44      0.47       377
weighted avg       1.00      0.88      0.94       377
```

OBSERVATION * After tuning, * accuracy, precision, recall & F1-Score remain same. * Type 1 is high.

##**DECISION TREE CLASSIFIER**

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

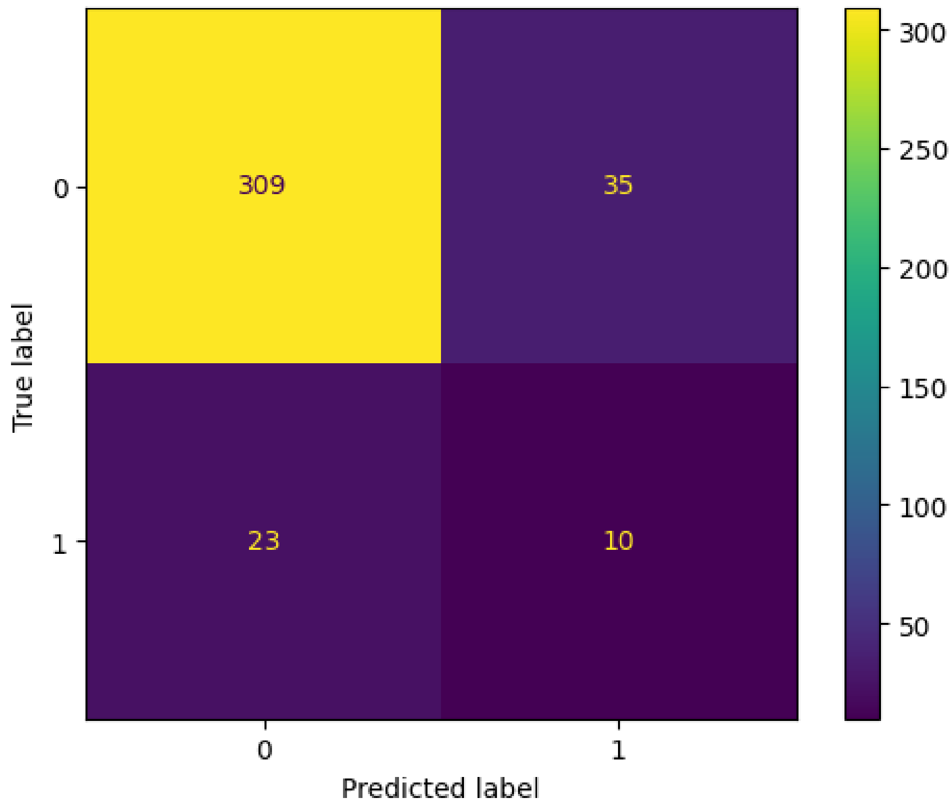
#####Baseline Decision Tree

[76]:
```
#Importing DecisionTreeClassifier from sklearn.
from sklearn.tree import DecisionTreeClassifier
dec_tree = DecisionTreeClassifier()

#Fitting the training independent and dependent variable to get trained.
dec_tree.fit(X_train,y_train)

#Predicting the result foe the test dataset "X_test".
y_pred_dec = dec_tree.predict(X_test)
```

[77]:
```
#Calculating the accuracy for the predicted and actual dataset.
dec_tree_untune = accuracy_score(y_pred_dec, y_test)*100
accuracy_score(y_pred_dec, y_test)*100
```

[77]:
```
84.61538461538461
```

[78]:
```
#Calculating for the confusion matrix.
dt = confusion_matrix(y_pred_dec, y_test, labels= dec_tree.classes_)
disp2 = ConfusionMatrixDisplay(confusion_matrix=dt, display_labels=dec_tree.
↪classes_)
disp2.plot()
plt.show()
```



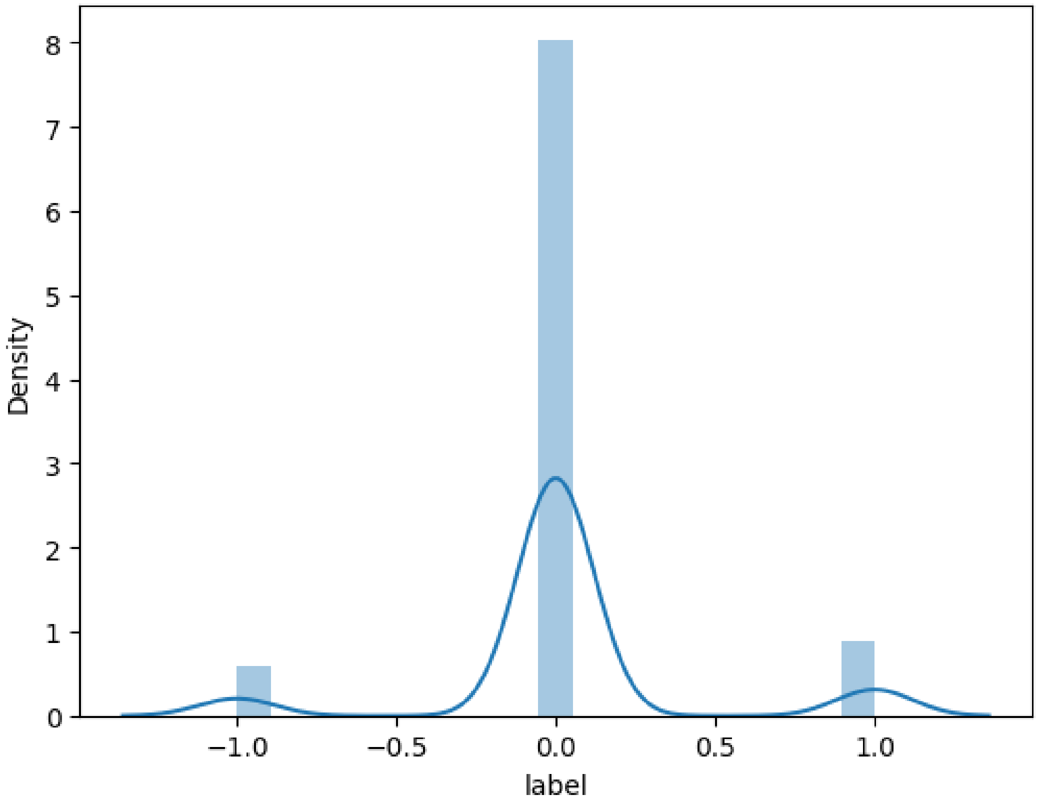
[79]:
```
#Importing accuracy_score, confusion_matrix, classification_report from sklearn.
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
print("Classification Report of Logistic Regression Model :␣
 ↪\n",classification_report(y_pred_dec, y_test))
```

```
Classification Report of Logistic Regression Model :
               precision    recall  f1-score   support

           0       0.93      0.90      0.91       344
           1       0.22      0.30      0.26        33

    accuracy                           0.85       377
   macro avg       0.58      0.60      0.59       377
weighted avg       0.87      0.85      0.86       377
```

OBSERVATIONS * Model is predicting the following:- * TP - "309" people get approval of credit card. * TN - "10" people not get approval of credit card. * FP - "35" people not get approval of credit card but model predicted they will get. * FN - "23" people get approval of credit card but model predicted they will not get.

[80]:
```
#Plotting the test - predicted graph.
sns.distplot(y_test - y_pred_dec)
```

[80]:
```
<Axes: xlabel='label', ylabel='Density'>
```



OBSERVATION * It follows Normal Distribution Curve. * There are less noise(error) at 1 & -1(value).

#####SMOTE Decision Tree

[81]:
```
dt_smote = DecisionTreeClassifier()
dt_smote.fit(X_train_sm, y_train_sm)

#Predicting the result foe the test dataset "X_test".
dt_smotee = dt_smote.predict(X_test)

print("Accuracy of Decision Tree Model : {}%".
 ↪format(round(accuracy_score(dt_smotee, y_test)*100)))
dt_smote_score = accuracy_score(dt_smotee, y_test)*100
#Calculating for the confusion matrix.
confusion_matrix(dt_smotee, y_test)
```

```
Accuracy of Decision Tree Model : 87%
```

[81]:
```
array([[302,  19],
       [ 30,  26]])
```

OBSERAVTION :- * Type 1 and Type 2 erro is less. * Accuracy Improved by 2%.

#####Hyper Parameter Tuning for Decision Tree Classifier

[82]:
```
#Importing GridSearchCV from sklearn
from sklearn.model_selection import GridSearchCV
parameter_dtc = {"criterion" : ["gini", "entropy", "log_loss"],
                 "splitter" : ["best", "random"],
                 "max_depth" : [1,2,3,4,5],
                 "max_features" : ["auto", "sqrt", "log2"],
                 "ccp_alpha" : [1,2,3,4,5,6,7]}
Classfier_dtc = GridSearchCV(dec_tree,param_grid=parameter_dtc,␣
 ↪scoring="accuracy", cv=10)
```

- Criterion for impurity of feature will be selected based on - “gini”, “entropy” or “log loss”
- Splitter use for best split after every feature selected b Information Gain.
- Max_depth use for how much depth we want to go in dividing the feature for purity.
- min_samples_split use for the minimum number of samples for each split.
- min_samples_leaf use for the minimum number of samples for each node.
- max_feature is the number of features to consider when looking for the best split.
- max_leaf_nodes use to control how much leaf nodes we required.,
- ccp_alpha - Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen.

[83]:
```
#Fitting the training independent and dependent variable to get trained.
Classfier_dtc.fit(X_train, y_train)
```

[83]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'ccp_alpha': [1, 2, 3, 4, 5, 6, 7],
                         'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [1, 2, 3, 4, 5],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'splitter': ['best', 'random']},
             scoring='accuracy')
```

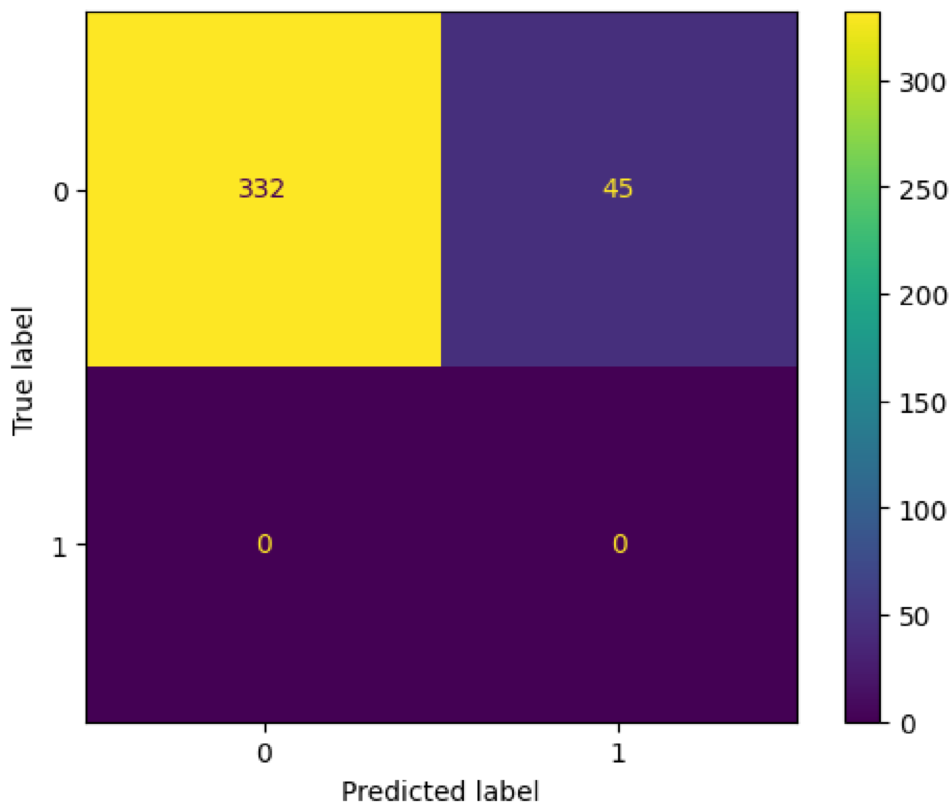
[84]:
```
#Getting the best parameter from gridsearchcv
print(Classfier_dtc.best_params_)

#getting the best score for training the model.
print(Classfier_dtc.best_score_)

#Predicting the result foe the test dataset "X_test".
y_pred_tune_dtc = Classfier_dtc.predict(X_test)
```

```
{'ccp_alpha': 1, 'criterion': 'gini', 'max_depth': 1, 'max_features': 'auto',
'splitter': 'best'}
0.8929203539823009
```

[85]:
```python
#Calculating the accuracy for the predicted and actual dataset.
dec_tree_tune = accuracy_score(y_pred_tune_dtc, y_test)*100
accuracy_score(y_pred_tune_dtc, y_test)*100
```

[85]: 88.06366047745358

[86]:
```python
#Calculating for the confusion matrix.
dt_tune = confusion_matrix(y_pred_tune_dtc, y_test, labels= Classfier_dtc.
 ↪classes_)
disp3 = ConfusionMatrixDisplay(confusion_matrix=dt_tune,␣
 ↪display_labels=Classfier_dtc.classes_)
disp3.plot()
plt.show()
```



[87]:
```python
#Getting the Classification report for precision, recall & f1-score value.
print(classification_report(y_pred_tune_dtc, y_test))
```

```
              precision    recall  f1-score   support
```

```
           0       1.00      0.88      0.94       377
           1       0.00      0.00      0.00         0

    accuracy                           0.88       377
   macro avg       0.50      0.44      0.47       377
weighted avg       1.00      0.88      0.94       377
```

OBSERVATION :- * Accuracy Improved by 4% from the baseline model and 1% by SMOTE Model. * Type 1 Error increased and Type 2 Error decreased.

##**KNN**

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called **a lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

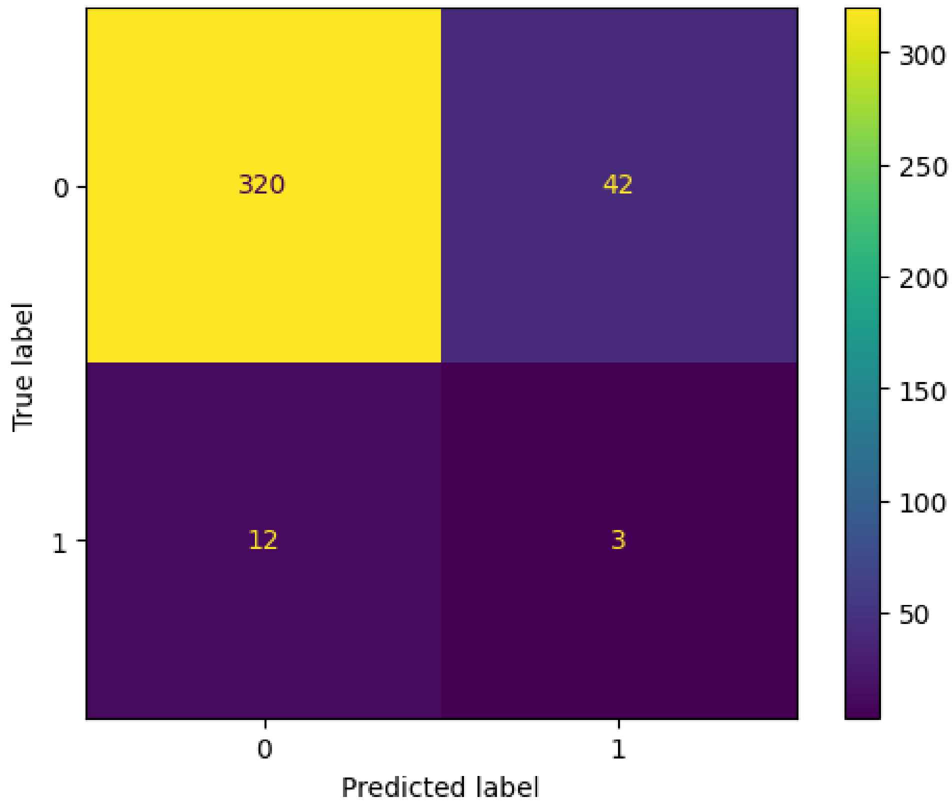
#####Baseline KNN

[88]:
```
#Importing KNeighborsClassifier from sklearn.
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()

#Fitting the training independent and dependent variable to get trained.
knn.fit(X_train,y_train)

#Predicting the result foe the test dataset "X_test".
y_pred_knn = knn.predict(X_test)
```

[89]:
```
#Calculating the accuracy for the predicted and actual dataset.
knn_untune = accuracy_score(y_pred_knn, y_test)*100
accuracy_score(y_pred_knn, y_test)*100
```

[89]: 85.6763925729443

[90]:
```
#Calculating for the confusion matrix.
knn_cm = confusion_matrix(y_pred_knn, y_test, labels= knn.classes_)
disp4 = ConfusionMatrixDisplay(confusion_matrix=knn_cm, display_labels=knn.
 ↪classes_)
disp4.plot()
plt.show()
```



OBSERVATIONS * Model is predicting the following:- * TP - "320" people get approval of credit card. * TN - "3" people not get approval of credit card. * FP - "42" people not get approval of credit card but model predicted they will get. * FN - "12" people get approval of credit card but model predicted they will not get.

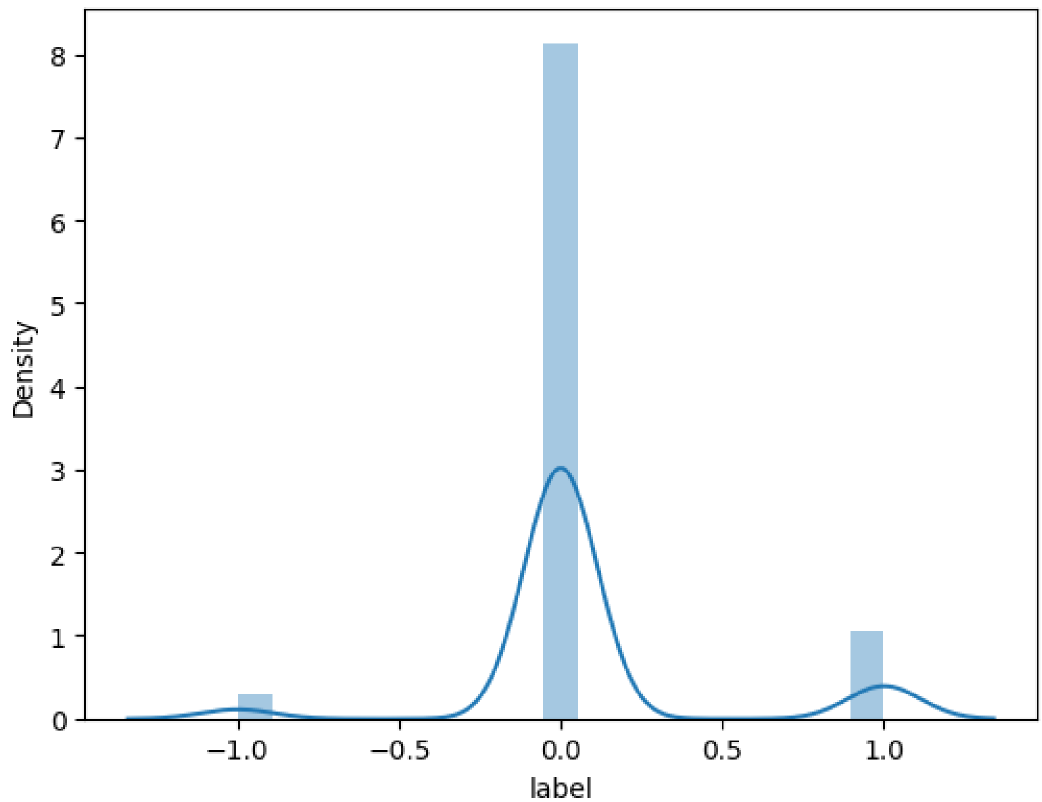
[91]:
```
#Importing accuracy_score, confusion_matrix, classification_report from sklearn.
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
print("Classification Report of Logistic Regression Model :␣
 ↪\n",classification_report(y_pred_knn, y_test))
```

```
Classification Report of Logistic Regression Model :
               precision    recall  f1-score   support

           0       0.96      0.88      0.92       362
```

```
           1       0.07      0.20      0.10        15

    accuracy                           0.86       377
   macro avg       0.52      0.54      0.51       377
weighted avg       0.93      0.86      0.89       377
```

[92]:
```
#Plotting the test - predicted graph.
sns.distplot(y_test - y_pred_knn)
```

[92]:
```
<Axes: xlabel='label', ylabel='Density'>
```



OBSERVATION * It follows Normal Distribution Curve. * There are less noise(error) at 1(value).

#####SMOTE KNN

[93]:
```
knn_smote = KNeighborsClassifier()
knn_smote.fit(X_train_sm, y_train_sm)

#Predicting the result foe the test dataset "X_test".
knn_smotee = knn_smote.predict(X_test)
```

```
print("Accuracy of Decision Tree Model : {}%".
 ↪format(round(accuracy_score(knn_smotee, y_test)*100)))
knn_smote_score = accuracy_score(knn_smotee, y_test)*100
#Calculating for the confusion matrix.
confusion_matrix(knn_smotee, y_test)
```

```
Accuracy of Decision Tree Model : 78%
```

```
[93]: array([[267,  19],
             [ 65,  26]])
```

OBSERVATION :- * SMOTE Technique doesn't help in KNN. * Error Increased and accuracy moved down.

#####Hyper Parameter Tuning for KNN

```
[94]: #Importing GridSearchCV from sklearn
from sklearn.model_selection import GridSearchCV
parameter_knn = {"n_neighbors" : list(np.arange(1,30,1)),
                 "weights" : ["uniform", "distance"],
                 "algorithm" : ["auto", "ball_tree", "kd_tree", "brute"]}
Classfier_knn = GridSearchCV(knn,param_grid=parameter_knn, scoring="accuracy",␣
 ↪cv=10)
```

```
[95]: #Fitting the training independent and dependent variable to get trained.
Classfier_knn.fit(X_train, y_train)
```

```
[95]: GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                         13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                                         23, 24, 25, 26, 27, 28, 29],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

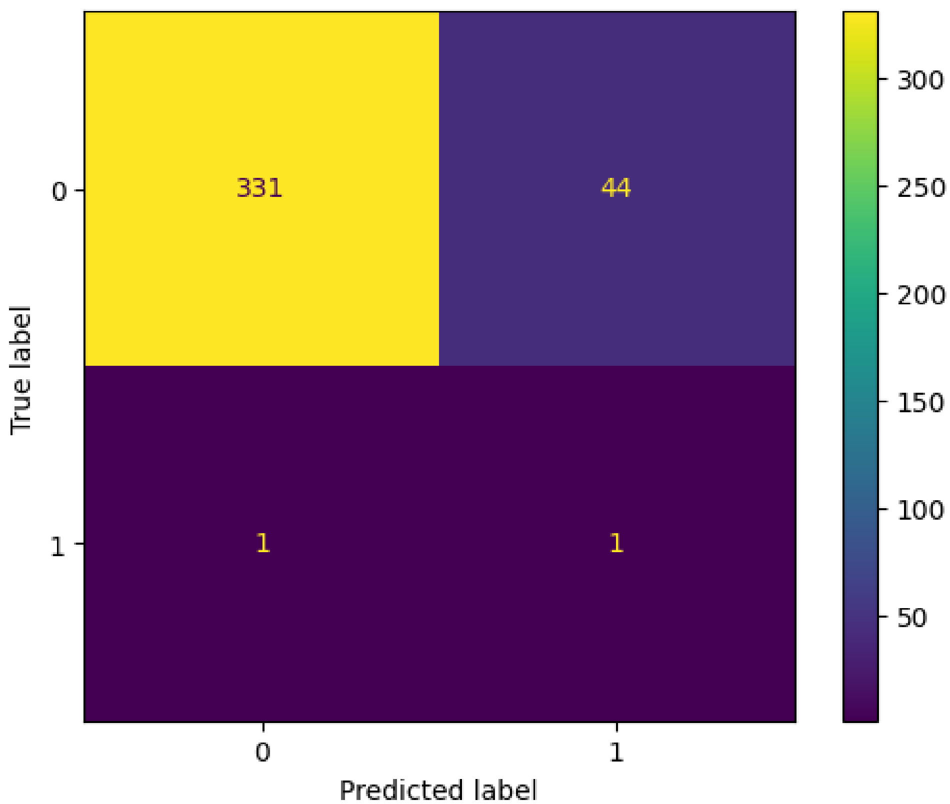
```
[96]: #Getting the best parameter from gridsearchcv.
print(Classfier_knn.best_params_)

#getting the best score for training the model.
print(Classfier_knn.best_score_)

#Predicting the result foe the test dataset "X_test".
y_pred_tune_knn = Classfier_knn.predict(X_test)
```

```
{'algorithm': 'auto', 'n_neighbors': 15, 'weights': 'distance'}
0.9185840707964601
```

[97]:
```
#Calculating the accuracy for the predicted and actual dataset.
knn_tune = accuracy_score(y_pred_tune_knn, y_test)*100
accuracy_score(y_pred_tune_knn, y_test)*100
```

[97]: 88.06366047745358

[98]:
```
#Calculating for the confusion matrix.
knn_tune_cm = confusion_matrix(y_pred_tune_knn, y_test, labels= Classfier_knn.
 ↪classes_)
disp5 = ConfusionMatrixDisplay(confusion_matrix=knn_tune_cm,␣
 ↪display_labels=Classfier_knn.classes_)
disp5.plot()
plt.show()
```



OBSERVATION :- * Type 1 Error Increased by 2 value and Type 2 Error Reduced by 11 values. * Improved accuracy by 2%.

[99]:
```
#Getting the Classification report for precision, recall & f1-score value.
print(classification_report(y_pred_tune_knn, y_test))
```

```
              precision    recall  f1-score   support
```

```
           0       1.00      0.88      0.94       375
           1       0.02      0.50      0.04         2

    accuracy                           0.88       377
   macro avg       0.51      0.69      0.49       377
weighted avg       0.99      0.88      0.93       377
```

##**RANDOM FOREST CLASSFIER**

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

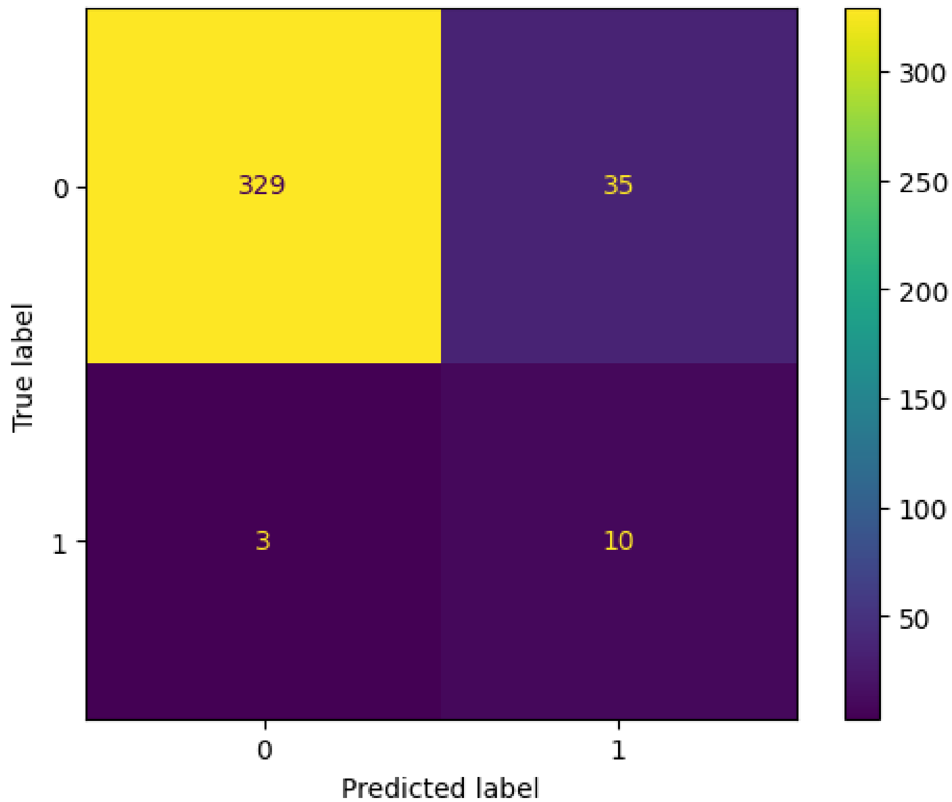
#####Baseline Random Forest

[100]:
```
#Importing RandomForestClassifier from sklearn.
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier()

#Fitting the training independent and dependent variable to get trained.
random_forest.fit(X_train,y_train)

#Predicting the result foe the test dataset "X_test".
y_pred_rfc = random_forest.predict(X_test)
```

[101]:
```
#Calculating the accuracy for the predicted and actual dataset.
random_forest_untune = accuracy_score(y_pred_rfc, y_test)*100
accuracy_score(y_pred_rfc, y_test)*100
```

[101]: 89.92042440318302

[102]:
```
#Calculating for the confusion matrix.
rf_cm = confusion_matrix(y_pred_rfc, y_test, labels= random_forest.classes_)
disp6 = ConfusionMatrixDisplay(confusion_matrix=rf_cm,␣
 ↪display_labels=random_forest.classes_)
disp6.plot()
plt.show()
```



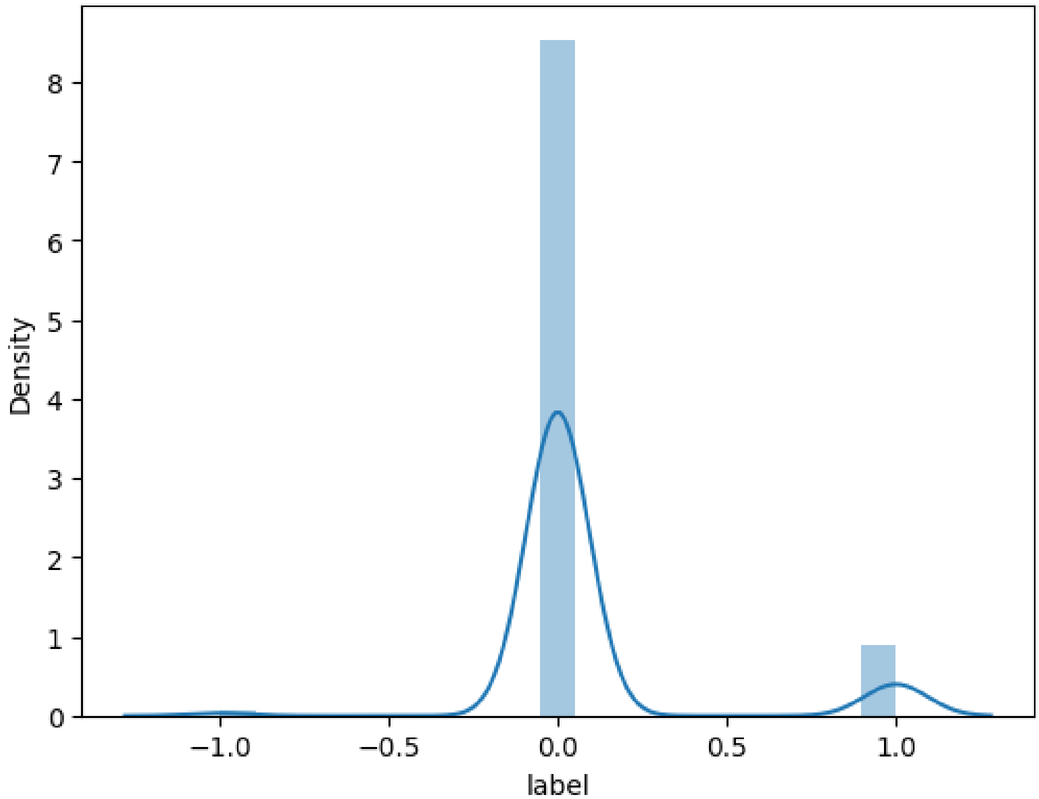
OBSERVATIONS * Model is predicting the following:- * TP - "329" people get approval of credit card. * TN - "10" people not get approval of credit card. * FP - "35" people not get approval of credit card but model predicted they will get. * FN - "3" people get approval of credit card but model predicted they will not get.

[103]:
```
#Importing accuracy_score, confusion_matrix, classification_report from sklearn.
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
print("Classification Report of Logistic Regression Model :␣
 ↪\n",classification_report(y_pred_rfc, y_test))
```

```
Classification Report of Logistic Regression Model :
               precision    recall  f1-score   support

           0       0.99      0.90      0.95       364
           1       0.22      0.77      0.34        13

    accuracy                           0.90       377
   macro avg       0.61      0.84      0.65       377
weighted avg       0.96      0.90      0.92       377
```

[104]:
```
sns.distplot(y_test - y_pred_rfc)
```

[104]: <Axes: xlabel='label', ylabel='Density'>



OBSERVATION * It follows Normal Distribution Curve. * There are less noise(error) at 1(value).

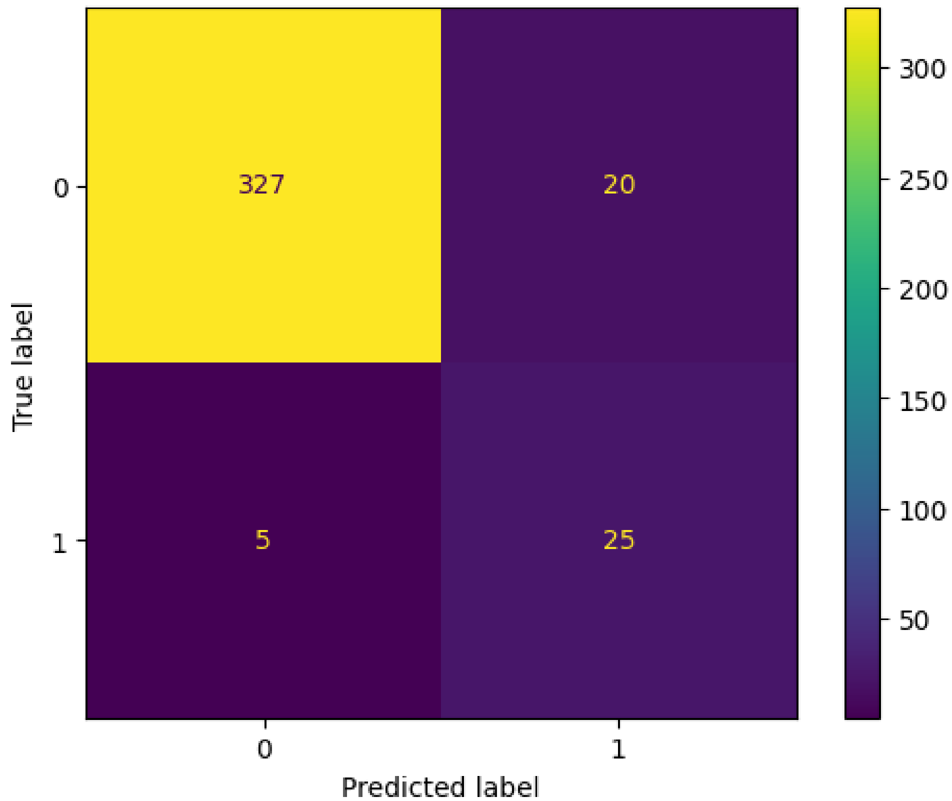
#####SMOTE Random Forest

[105]:
```
rf_smote = RandomForestClassifier()
rf_smote.fit(X_train_sm, y_train_sm)

#Predicting the result foe the test dataset "X_test".
rf_smotee = rf_smote.predict(X_test)

print("Accuracy of Decision Tree Model : {}%".format((accuracy_score(rf_smotee,␣
↪y_test)*100)))
rf_smote_score = accuracy_score(rf_smotee, y_test)*100
```

```
Accuracy of Decision Tree Model : 93.36870026525199%
```

[106]:
```
#Calculating for the confusion matrix.
rf_cm_smote = confusion_matrix(rf_smotee, y_test, labels= rf_smote.classes_)
```

```
disp6 = ConfusionMatrixDisplay(confusion_matrix=rf_cm_smote,␣
↪display_labels=rf_smote.classes_)
disp6.plot()
plt.show()
```



[107]:
```
print("Classification Report of Logistic Regression Model :␣
↪\n",classification_report(rf_smotee, y_test))
```

```
Classification Report of Logistic Regression Model :
               precision    recall  f1-score   support

           0       0.98      0.94      0.96       347
           1       0.56      0.83      0.67        30

    accuracy                           0.93       377
   macro avg       0.77      0.89      0.81       377
weighted avg       0.95      0.93      0.94       377
```

OBSERVATION :- * Reduced the False Positive by 43% that is Credit Card Approval is less for Model for people who are not qualified. * Accuracy Improved by 3% from 90% to 93% with good precision and recall.

#####Hyper Parameter Tuning for Random Forest Classifier

[108]:
```
#Importing GridSearchCV from sklearn.
from sklearn.model_selection import GridSearchCV
parameter_rfc = {"n_estimators" : list(np.arange(10,15,1)),
                 "criterion" : ["gini", "entropy", "log_loss"],
                 "max_depth" : list(np.arange(0,14))}
Classfier_rfc = GridSearchCV(random_forest, param_grid=parameter_rfc,␣
 ↪scoring="accuracy", cv=10)
```

[109]:
```
#Fitting the training independent and dependent variable to get trained.
Classfier_rfc.fit(X_train, y_train)
```

[109]:
```
GridSearchCV(cv=10, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                       13],
                         'n_estimators': [10, 11, 12, 13, 14]},
             scoring='accuracy')
```

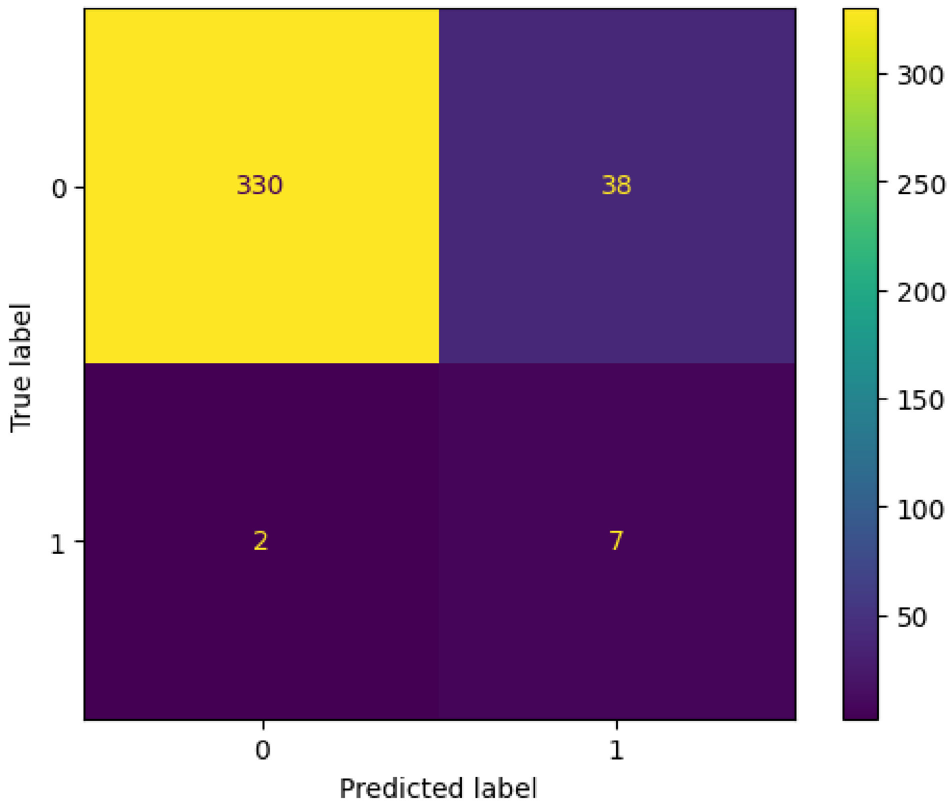
[110]:
```
#Getting the best parameter from gridsearchcv.
print(Classfier_rfc.best_params_)

#getting the best score for training the model.
print(Classfier_rfc.best_score_)

#Predicting the result foe the test dataset "X_test".
y_pred_tune_rfc = Classfier_rfc.predict(X_test)
```

```
{'criterion': 'gini', 'max_depth': 12, 'n_estimators': 14}
0.9168141592920355
```

[111]:
```
#Calculating the accuracy for the predicted and actual dataset.
random_forest_tune = accuracy_score(y_pred_tune_rfc, y_test)*100
accuracy_score(y_pred_tune_rfc, y_test)*100
```

[111]:
```
89.38992042440319
```

[112]:
```
#Calculating for the confusion matrix.
rf_tune_cm = confusion_matrix(y_pred_tune_rfc, y_test, labels= Classfier_rfc.
 ↪classes_)
disp7 = ConfusionMatrixDisplay(confusion_matrix=rf_tune_cm,␣
 ↪display_labels=Classfier_rfc.classes_)
disp7.plot()
plt.show()
```



[113]:
```
#Getting the Classification report for precision, recall & f1-score value.
print(classification_report(y_pred_tune_rfc, y_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.90      0.94       368
           1       0.16      0.78      0.26         9

    accuracy                           0.89       377
   macro avg       0.57      0.84      0.60       377
weighted avg       0.97      0.89      0.93       377
```

OBSERVATION :- * Tuned Model doesn't help for better accuracy.

##**XGBOOST**

#####Baseline XG Boost

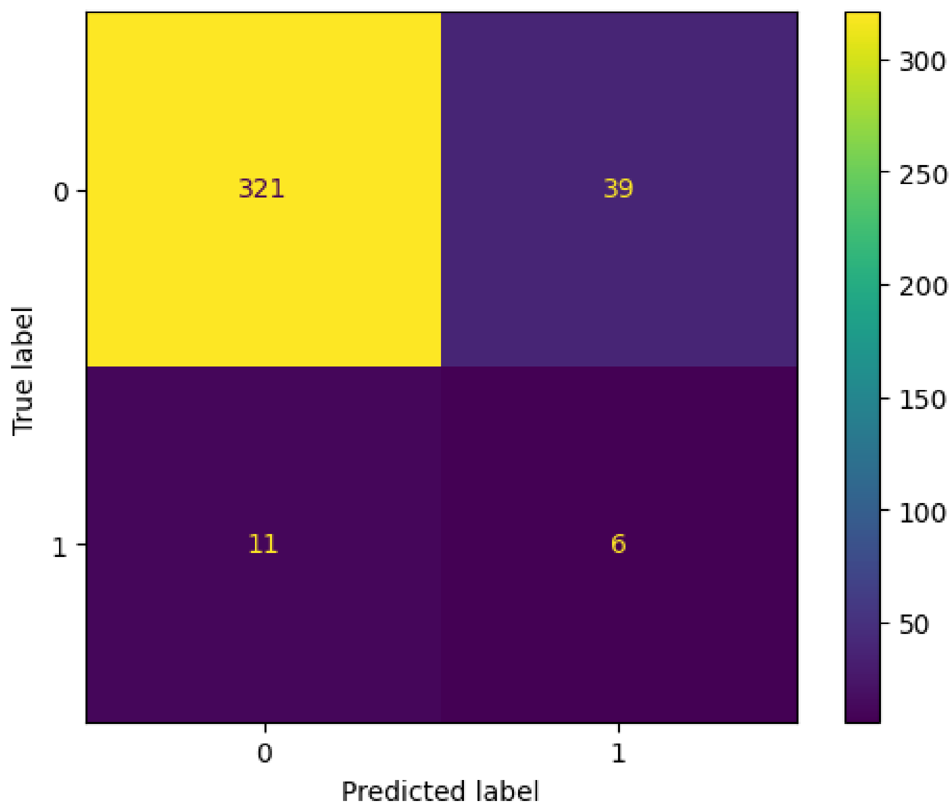
[114]:
```
#Importing XGBoost.
import xgboost as xgb
xgb_classifier = xgb.XGBClassifier()
```

```
#Fitting the training independent and dependent variable to get trained.
xgb_classifier.fit(X_train,y_train)

#Predicting the result foe the test dataset "X_test".
y_pred_xgb = xgb_classifier.predict(X_test)
```

[115]:
```
#Calculating the accuracy for the predicted and actual dataset.
accuracy_score(y_pred_xgb, y_test)*100
```

[115]: 86.73740053050398

[116]:
```
#Calculating for the confusion matrix.
xgb_cm = confusion_matrix(y_pred_xgb, y_test, labels= xgb_classifier.classes_)
disp8 = ConfusionMatrixDisplay(confusion_matrix=xgb_cm,␣
 ↪display_labels=xgb_classifier.classes_)
disp8.plot()
plt.show()
```



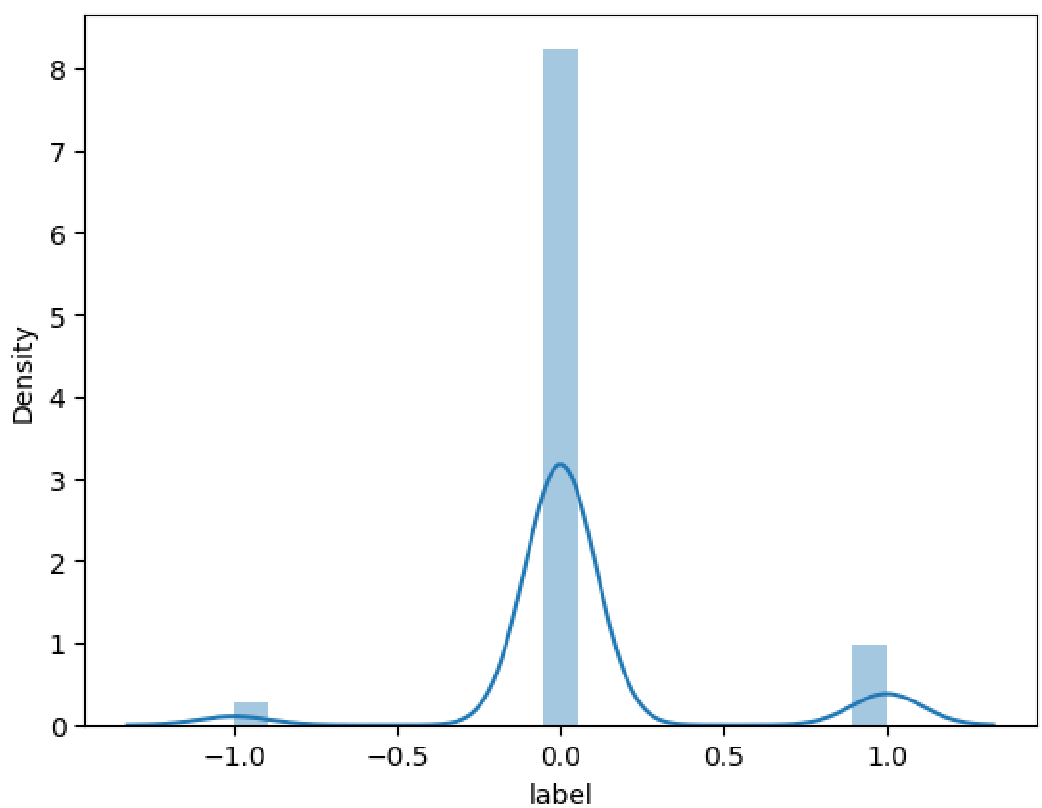
OBSERVATIONS * Model is predicting the following:- * TP - "321" people get approval of credit card. * TN - "6" people not get approval of credit card. * FP - "39" people not get approval of credit card but model predicted they will get. * FN - "11" people get approval of credit card but model predicted they will not get.

[117]:
```
#Importing accuracy_score, confusion_matrix, classification_report from sklearn.
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
print("Classification Report of Logistic Regression Model :␣
↪\n",classification_report(y_pred_xgb, y_test))
```

```
Classification Report of Logistic Regression Model :
              precision    recall  f1-score   support

           0       0.97      0.89      0.93       360
           1       0.13      0.35      0.19        17

    accuracy                           0.87       377
   macro avg       0.55      0.62      0.56       377
weighted avg       0.93      0.87      0.89       377
```

[118]:
```
#Plotting the test - predicted graph.
sns.distplot(y_test - y_pred_xgb)
```

[118]: <Axes: xlabel='label', ylabel='Density'>



OBSERVATION * It follows Normal Distribution Curve. * There are less noise(error) at 1(value).

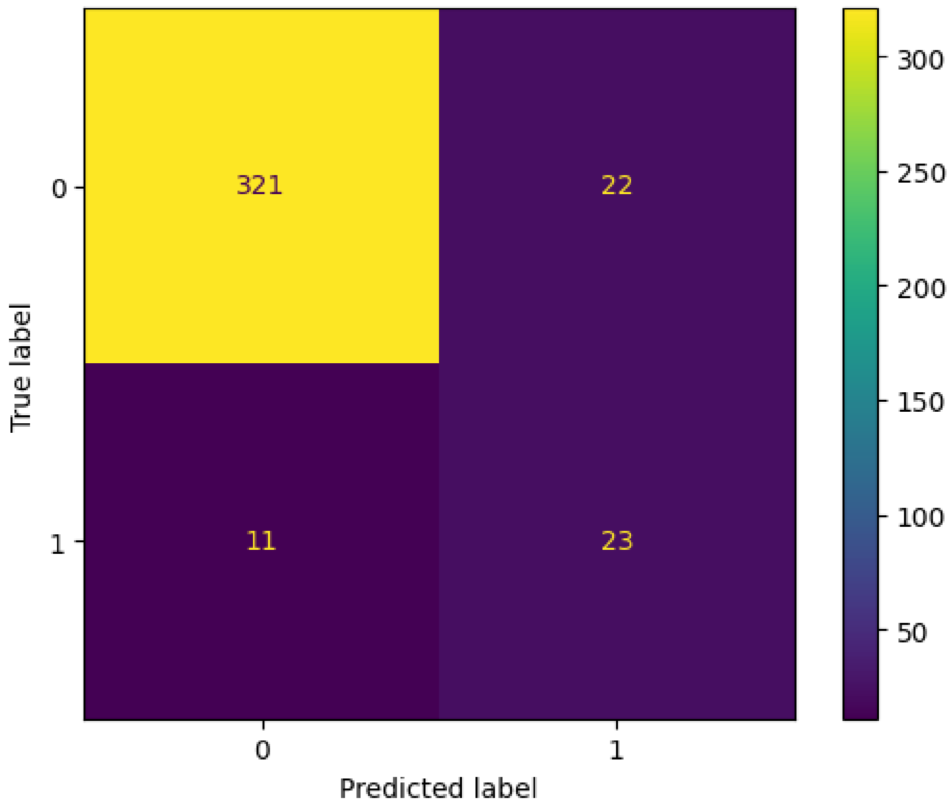
#####SMOTE XG Boost

[119]:
```
xgb_smote = xgb.XGBClassifier()
xgb_smote.fit(X_train_sm, y_train_sm)

#Predicting the result foe the test dataset "X_test".
xgb_smotee = xgb_smote.predict(X_test)

print("Accuracy of Decision Tree Model : {}%".
 ↪format(round(accuracy_score(xgb_smotee, y_test)*100)))
xgb_smote_score = accuracy_score(xgb_smotee, y_test)*100
#Calculating for the confusion matrix.
confusion_matrix(xgb_smotee, y_test)
```

```
Accuracy of Decision Tree Model : 91%
```

[119]:
```
array([[321,  22],
       [ 11,  23]])
```

[120]:
```
xgb_smote_cm = confusion_matrix(xgb_smotee, y_test, labels= xgb_smote.classes_)
disp9 = ConfusionMatrixDisplay(confusion_matrix=xgb_smote_cm,␣
 ↪display_labels=xgb_smote.classes_)
disp9.plot()
plt.show()
```

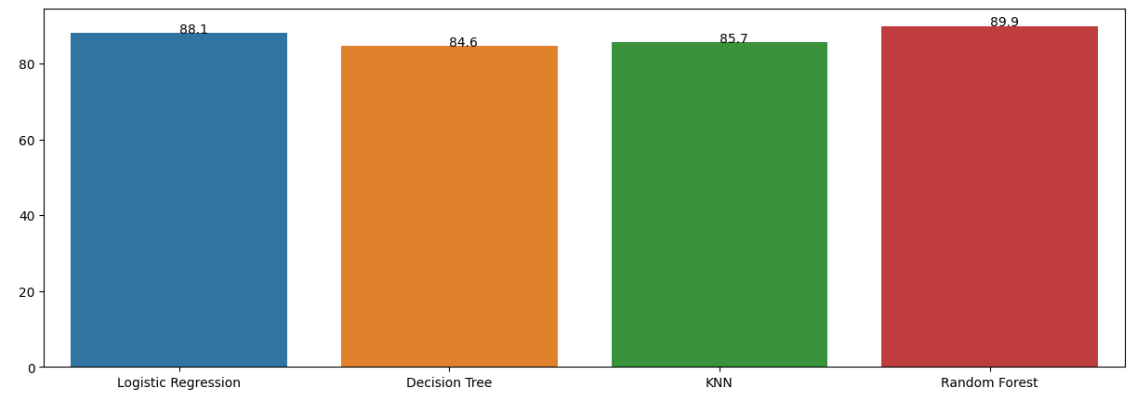


OBSERAVTION * SMOTE Model is better than Baseline Model. * It has better Precision, Recall and F1 Score. * Accuracy Improved by 4%.

#ACCURACY : GRAPH

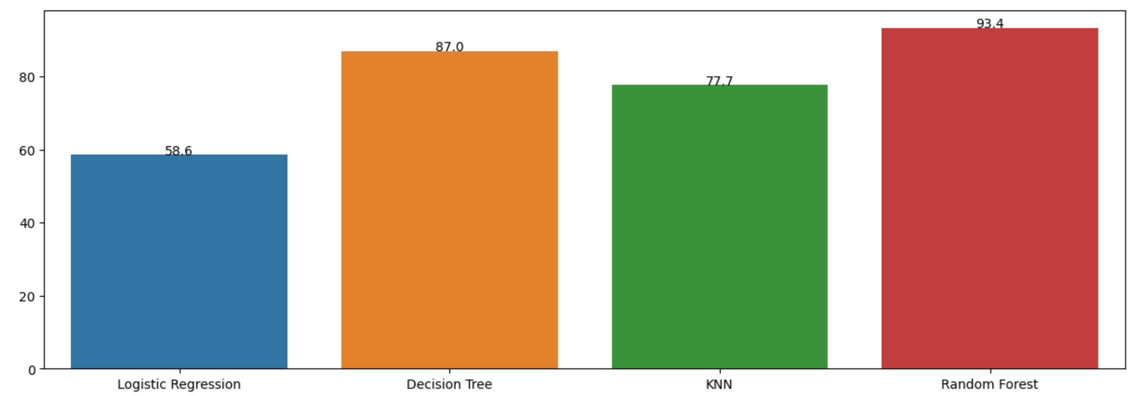
#####Baseline Model Graph

[121]:
```
untuned_algo = [Log_reg_untune, dec_tree_untune, knn_untune,␣
↪random_forest_untune]
Algo_value_untune = []
for i in untuned_algo:
  Algo_value_untune.append(round(i,1))
Algo = ["Logistic Regression", "Decision Tree" , "KNN", "Random Forest"]
plt.figure(figsize=(15,5))
def addlabels_untune(x,y):
    for i in range(len(x)):
        plt.text(i,y[i],y[i], horizontalalignment = "left")
sns.barplot(x= Algo, y = untuned_algo)
addlabels_untune(Algo, Algo_value_untune)
```



#####SMOTE Algorithm Graph

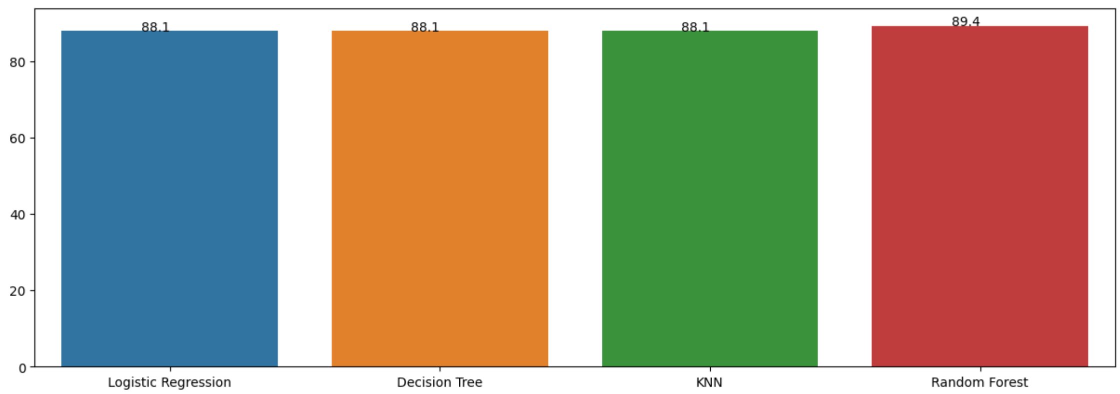
[122]:
```
untuned_algo_smote = [lg_smote_score, dt_smote_score, knn_smote_score,␣
 ↪rf_smote_score]
Algo_value_untune_smote = []
for i in untuned_algo_smote:
  Algo_value_untune_smote.append(round(i,1))
Algo_smote = ["Logistic Regression", "Decision Tree" , "KNN", "Random Forest"]
plt.figure(figsize=(15,5))
def addlabels_untune_smote(x,y):
    for i in range(len(x)):
        plt.text(i,y[i],y[i], horizontalalignment = "center")
sns.barplot(x= Algo_smote, y = untuned_algo_smote)
addlabels_untune_smote(Algo_smote, Algo_value_untune_smote)
```



#####Tuned Algorithm Graph

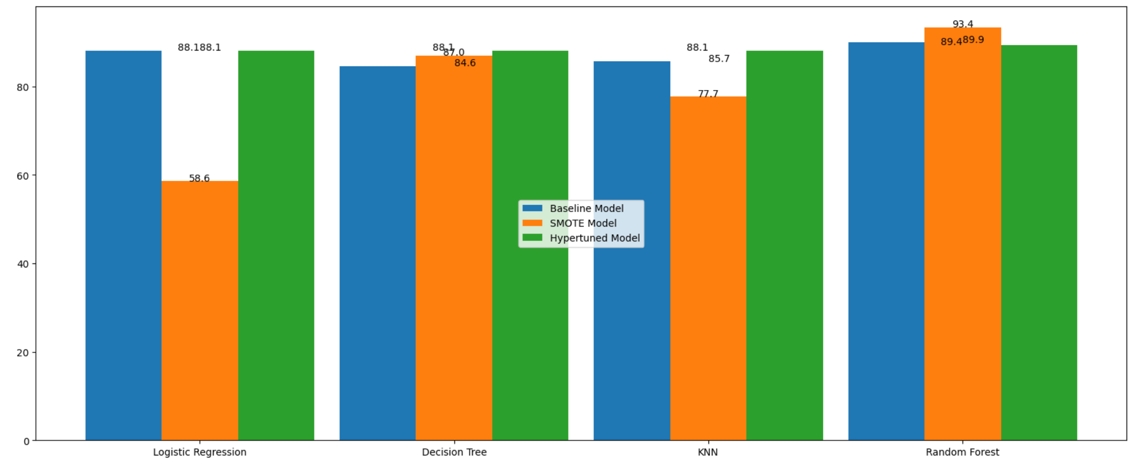
[123]:
```
tuned_algo = [Log_reg_tune, dec_tree_tune, knn_tune, random_forest_tune]
Algo_value_tune = []
```

```
for i in tuned_algo:
  Algo_value_tune.append(round(i,1))
Algo = ["Logistic Regression", "Decision Tree" , "KNN", "Random Forest"]
plt.figure(figsize=(15,5))
def addlabels_tune(x,y):
    for i in range(len(x)):
        plt.text(i,y[i],y[i], horizontalalignment = "right")
sns.barplot(x = Algo, y = Algo_value_tune)
addlabels_tune(Algo, Algo_value_tune)
```



#####Comparision

[124]:
```
plt.figure(figsize=(20,8))
X_label_algo = np.arange(len(Algo))
plt.bar(X_label_algo-0.3, untuned_algo, 0.3, label = "Baseline Model")
plt.bar(X_label_algo, untuned_algo_smote, 0.3, label = "SMOTE Model")
plt.bar(X_label_algo+0.3, tuned_algo, 0.3, label = "Hypertuned Model" )
plt.xticks(X_label_algo, Algo)
plt.legend(loc = "center")
addlabels_tune(Algo, Algo_value_tune)
addlabels_untune(Algo, Algo_value_untune)
addlabels_untune_smote(Algo_smote, Algo_value_untune_smote)
```



[124]:

#**CONCLUSION**

###**OBSERVATION** * From all the models, **Random Forest SMOTE Model** has better prediction.

- For **Random Forest "SMOTE" Model** :-
  - FP(False Positive) **reduced to 43%** from Baseline Model.
    - i.e, Approving Credit Card Error is reduced by 43% from Base Model.
  - Precision is **94.23%** which is good.
    - i.e, Correct Prediction Rate is 94.23%.
  - Recall is **98.5%** which is also good.
    - Out of all Credit Card Approval, 98.5% is positiviely approved.
  - F1-Score is **96.4%**, it is good.
    - It tell us how good our Precision and Recall is.Here It tells how Correctly our model predicted credit card approval from wrong approval.
  - Accuracy is **93.5%**.
    - It tells out of 100 people, 93 people credit card approval will be correct.

###**CONCLUSION** * From Random Forest, **Random Forest SMOTE Model gives better prediction and less error**. * **Random Forest Model give better Precision, Recall and Accuracy**.