LOAN APPROVAL ANALYSIS

In [163]:
```
# Importing the important libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

#tensorflow lib
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

In [164]:
```
# Loading the dataset
df = pd.read_json("C:/Users/Likhit Gaikwad/Downloads/loan_approval_d
df.head(10)
```

Out[164]:

	Id	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownership
0	1	1303834	23	3	single	rented	no
1	2	7574516	40	10	single	rented	no
2	3	3991815	66	4	married	rented	no
3	4	6256451	41	2	single	rented	yes
4	5	5768871	47	11	single	rented	no
5	6	6915937	64	0	single	rented	no
6	7	3954973	58	14	married	rented	no
7	8	1706172	33	2	single	rented	no
8	9	7566849	24	17	single	rented	yes
9	10	8964846	23	12	single	rented	no

Exploratory Data Analysis (EDA)

In [165]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 252000 entries, 0 to 251999
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   Id                 252000 non-null  int64 
 1   Income             252000 non-null  int64 
 2   Age                252000 non-null  int64 
 3   Experience         252000 non-null  int64 
 4   Married/Single     252000 non-null  object
 5   House_Ownership    252000 non-null  object
 6   Car_Ownership      252000 non-null  object
 7   Profession         252000 non-null  object
 8   CITY               252000 non-null  object
 9   STATE              252000 non-null  object
 10  CURRENT_JOB_YRS    252000 non-null  int64 
 11  CURRENT_HOUSE_YRS  252000 non-null  int64 
 12  Risk_Flag          252000 non-null  int64 
dtypes: int64(7), object(6)
memory usage: 26.9+ MB
```

In [166]:
```
df.describe()
```

Out[166]:

	Id	Income	Age	Experience	CURRENT_JOB_
count	252000.000000	2.520000e+05	252000.000000	252000.000000	252000.00
mean	126000.500000	4.997117e+06	49.954071	10.084437	6.33
std	72746.278255	2.878311e+06	17.063855	6.002590	3.64
min	1.000000	1.031000e+04	21.000000	0.000000	0.00
25%	63000.750000	2.503015e+06	35.000000	5.000000	3.00
50%	126000.500000	5.000694e+06	50.000000	10.000000	6.00
75%	189000.250000	7.477502e+06	65.000000	15.000000	9.00
max	252000.000000	9.999938e+06	79.000000	20.000000	14.00

In [167]:
```
# Finding Null values
df.isnull().sum()
```

Out[167]:
```
Id                   0
Income               0
Age                  0
Experience           0
Married/Single       0
House_Ownership      0
Car_Ownership        0
Profession           0
CITY                 0
STATE                0
CURRENT_JOB_YRS      0
CURRENT_HOUSE_YRS    0
Risk_Flag            0
dtype: int64
```

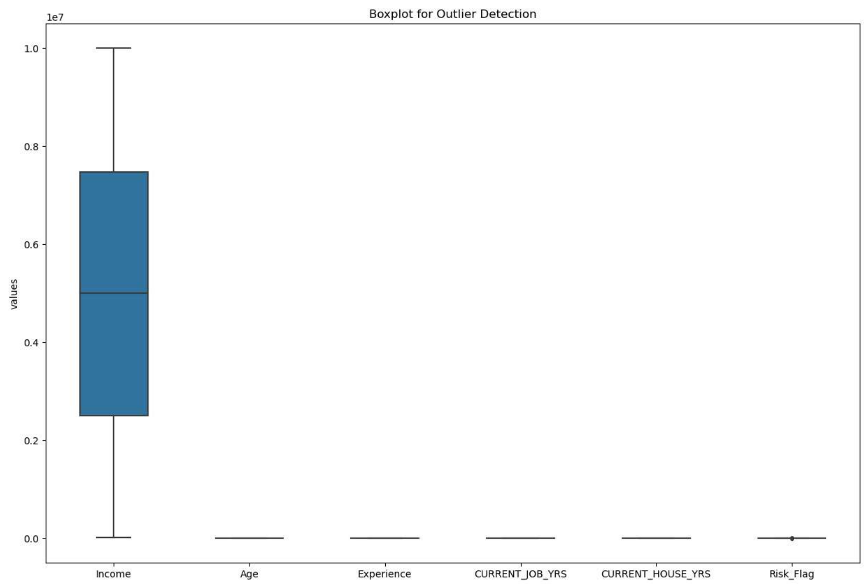
In [170]:
```
# Checking for duplicates in the data
duplicates = df[df.duplicated()]
print(duplicates)
```

```
Empty DataFrame
Columns: [Id, Income, Age, Experience, Married/Single, House_Owner
ship, Car_Ownership, Profession, CITY, STATE, CURRENT_JOB_YRS, CUR
RENT_HOUSE_YRS, Risk_Flag]
Index: []
```

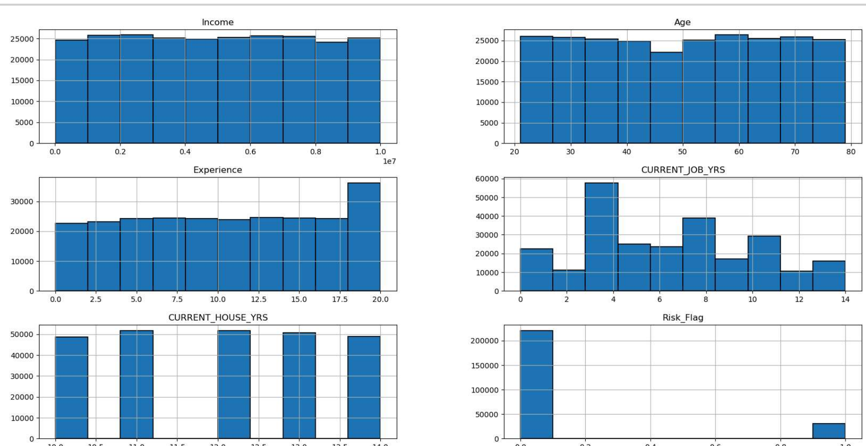
Data Visualization

In [171]:
```
# Removing the ID column
df = df.drop(columns=['Id'])
```

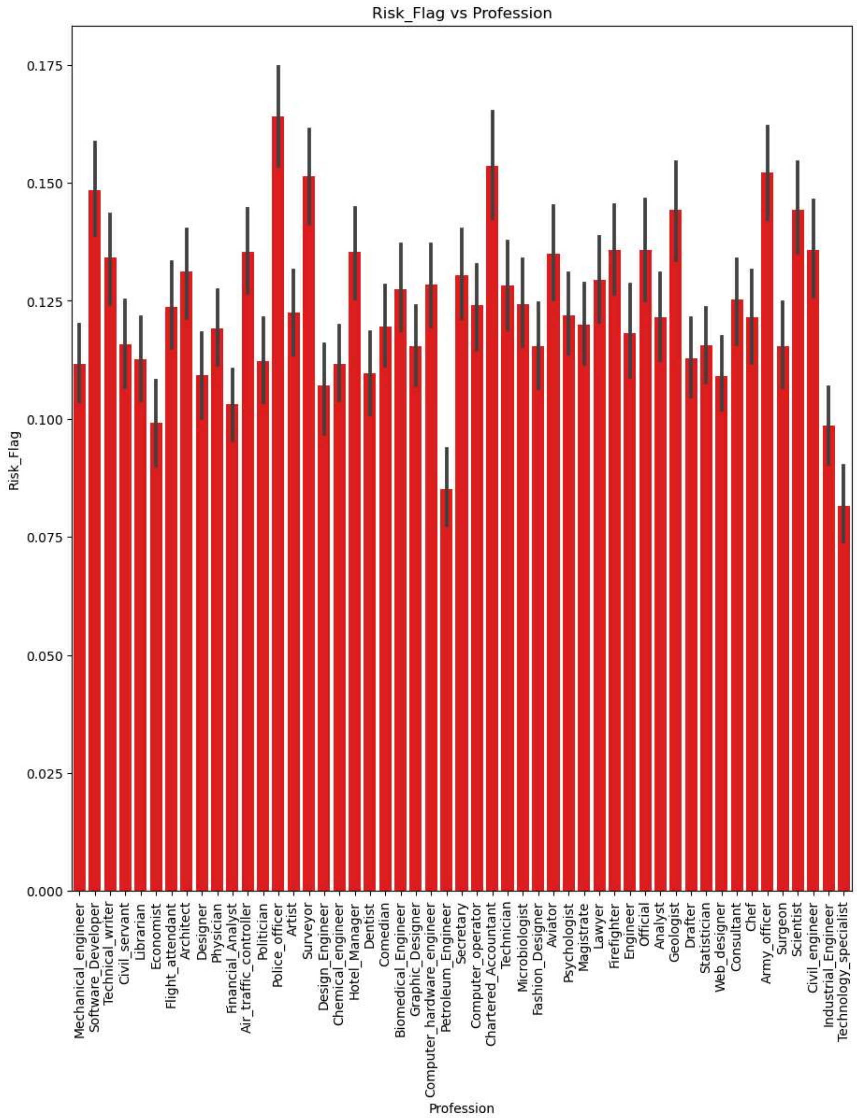
In [172]:
```
# Checking for outliers in the data
fig ,ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=df, width=0.5, ax=ax, fliersize=3)
ax.set_title('Boxplot for Outlier Detection')
ax.set_ylabel('values')
plt.show()
```



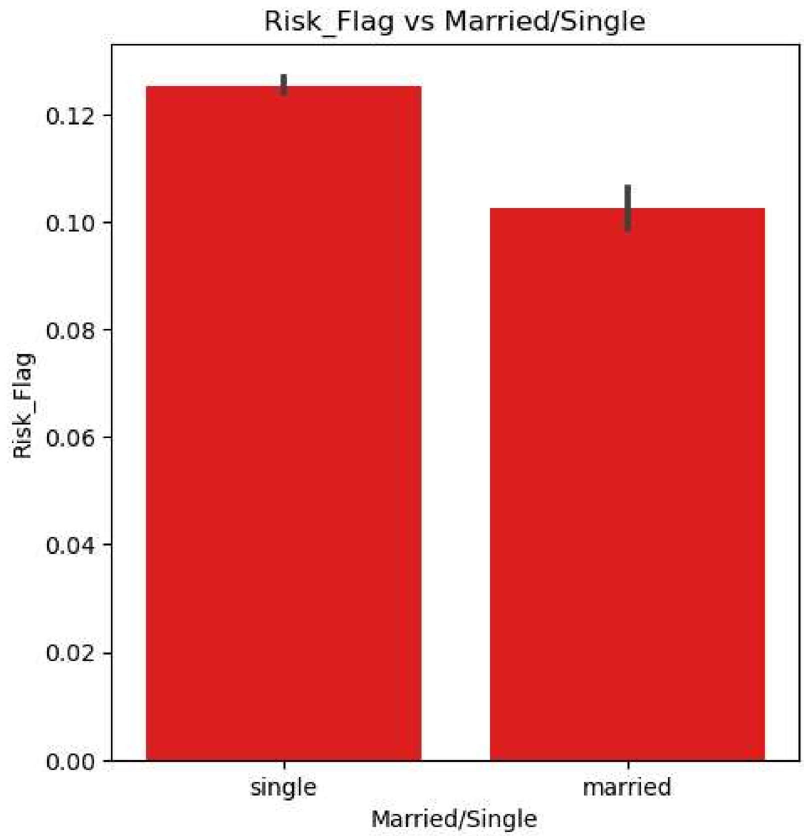
In [173]:
```
# Histogram plot on different variables
df.hist(edgecolor='black', linewidth=1.2, figsize=(20,10))
plt.show()
```



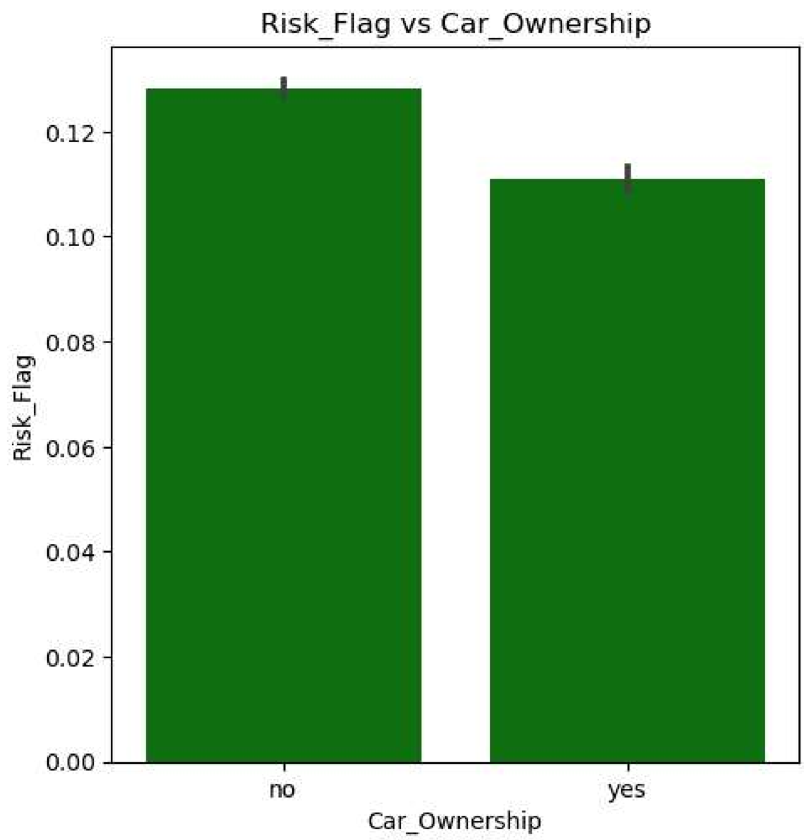
In [36]:
```
# Barplot between Profession and Risk_Flag
plt.figure(figsize=(10,10))
sns.barplot(x= df['Profession'],y=df['Risk_Flag'], color='red')
plt.tight_layout()
plt.title("Risk_Flag vs Profession")
plt.xticks(rotation=90)
plt.show()
```



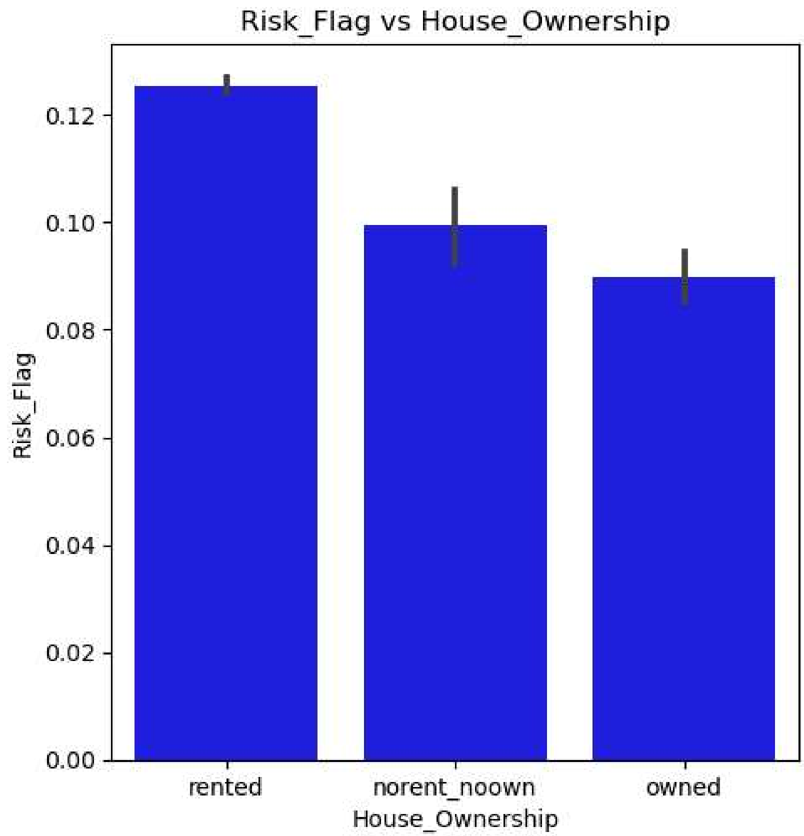
In [157]:
```
# Barplot between Marital status and Risk_Flag
plt.figure(figsize=(5,5))
sns.barplot(x=df['Married/Single'],y=df['Risk_Flag'], color='red')
plt.tight_layout()
plt.title("Risk_Flag vs Married/Single")
plt.show()
```



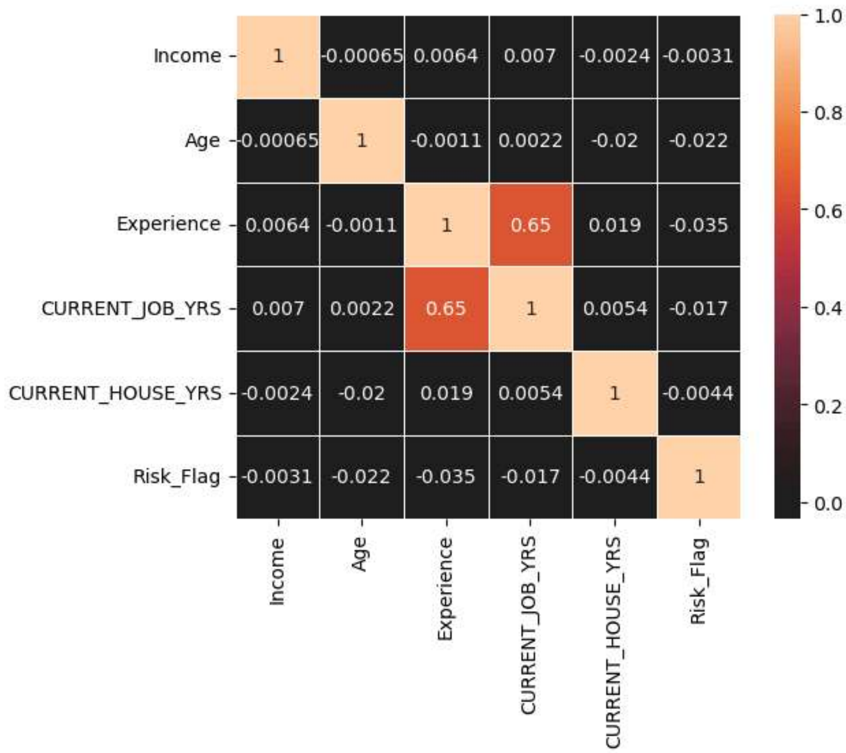
In [158]:
```
# Barplot between Car_Ownership and Risk_Flag
plt.figure(figsize=(5,5))
sns.barplot(x=df['Car_Ownership'],y=df['Risk_Flag'], color='green')
plt.tight_layout()
plt.title("Risk_Flag vs Car_Ownership")
plt.show()
```



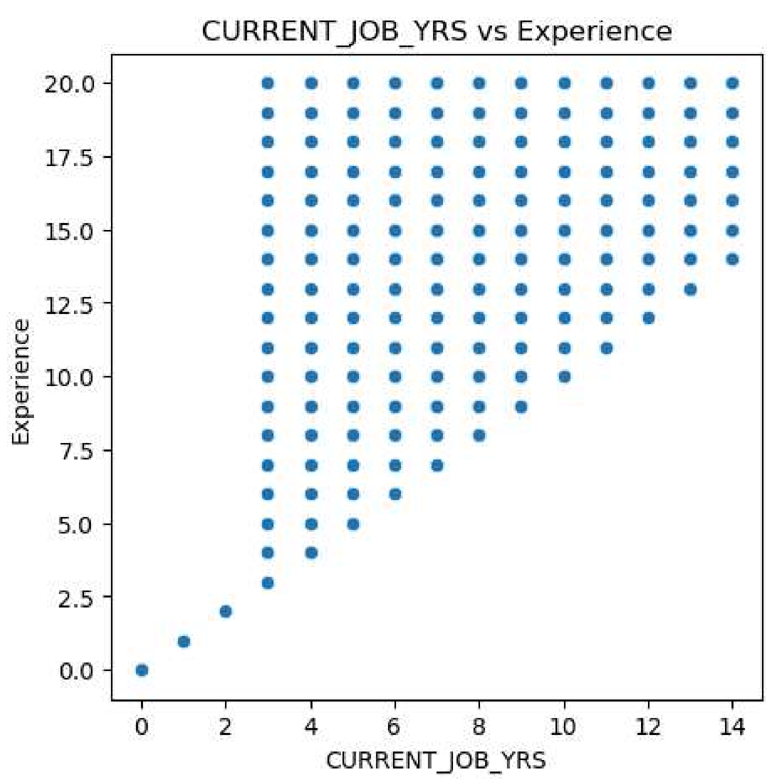
In [159]:
```
# Barplot between House_Ownership and Risk_Flag
plt.figure(figsize=(5,5))
sns.barplot(x=df['House_Ownership'],y=df['Risk_Flag'], color='blue')
plt.tight_layout()
plt.title("Risk_Flag vs House_Ownership")
plt.show()
```



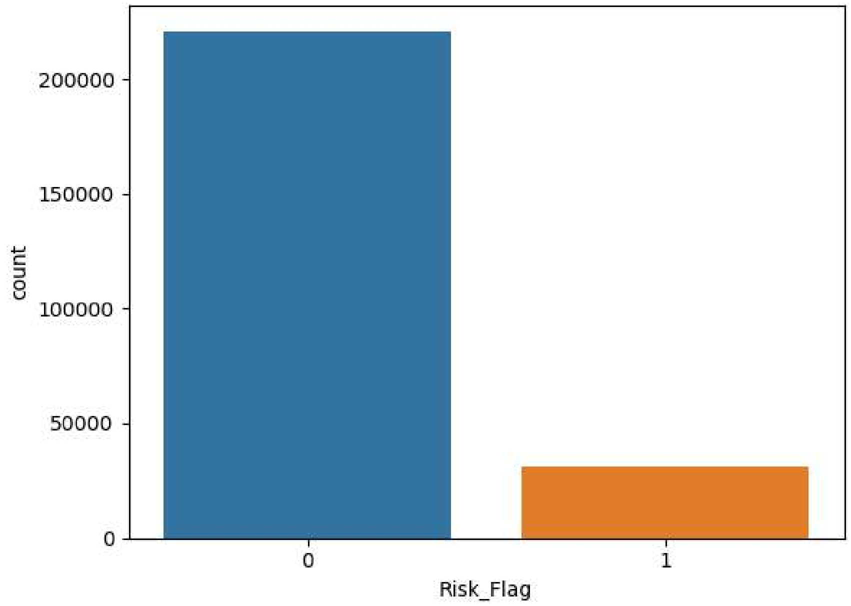
In [153]:
```
# Identifying correlation between variables
corr = df.corr()
sns.heatmap(corr, annot=True, center=0, square=True, linewidth=0.5)
plt.show()
```



In [156]:
```
# Scatterplot between CURRENT_JOB_YRS and Experience since they are
plt.figure(figsize=(5,5))
plt.title("CURRENT_JOB_YRS vs Experience")
sns.scatterplot(y='Experience', x='CURRENT_JOB_YRS', data=df)
plt.show()
```



In [154]:
```
# Visualizing the target variable
sns.countplot(x='Risk_Flag', data=df)
plt.show()
```



In [40]:
```
# There is a large imbalance in the target variable
# Making the target variable balanced

# Separating the classes
df_majority = df[df['Risk_Flag']==0]
df_minority = df[df['Risk_Flag']==1]

# Downsampling the majority class
df_majority_downsampled = df_majority.sample(len(df_minority), rando

# Combining the downsampled majority and minority classes
df_new = pd.concat([df_majority_downsampled, df_minority])

# Counting the Target variable
print(df_new['Risk_Flag'].value_counts())
```

```
0    30996
1    30996
Name: Risk_Flag, dtype: int64
```

In [82]:
```
# Removing the target variable
X_old = df_new.drop(columns=['Risk_Flag'])
X_old
```

Out[82]:

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownershi
114032	8823581	27	15	single	rented	n
213746	5157518	23	6	single	rented	n
163964	4041403	47	19	single	rented	ye
155854	8622588	45	18	single	owned	n
101715	3803419	61	16	married	rented	n
...	...	...	...	...	...	.
251973	1244622	35	15	single	rented	n
251977	1330613	63	19	single	rented	n
251981	1796713	47	2	single	rented	n
251982	3182290	52	2	single	rented	n
251993	8141027	60	10	single	rented	n

61992 rows × 11 columns

In [83]:
```
nverting the categorical variables to dummy/indicator variables
igning data to the X and y variables
pd.get_dummies(X_old, columns=['Married/Single','House_Ownership', '(
df_new['Risk_Flag']
```

In [43]:
```
# Splitting the train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
```

In [44]:
```
# Scaling the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

In [45]:
```
X_train.shape
```

Out[45]: (43394, 403)

Model Performance

In [190]:
```
# Building the model

model = Sequential()

model.add(Dense(800, activation='relu', kernel_initializer='he_unifo
model.add(Dropout(0.2))

model.add(Dense(400, activation='sigmoid')) #hidden layer 2
model.add(Dropout(0.5))

model.add(Dense(200, activation = 'relu', kernel_initializer='he_nor
model.add(Dropout(0.1))

model.add(Dense(100, activation = 'sigmoid')) #hidden layer 4
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid')) # output layer
```

In [191]:
```
# Compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
```

In [192]:
```
# Training the data
model.fit(X_train, y_train, epochs=30, batch_size=30)
```

```
Epoch 1/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.5550 - l
oss: 0.6865
Epoch 2/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.6958 - lo
ss: 0.5823
Epoch 3/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.7928 - lo
ss: 0.4652
Epoch 4/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.8285 - lo
ss: 0.4132
Epoch 5/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8486 - lo
ss: 0.3812
Epoch 6/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8597 - lo
ss: 0.3611
Epoch 7/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8705 - lo
ss: 0.3360
Epoch 8/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.8787 - lo
ss: 0.3234
Epoch 9/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.8837 - lo
ss: 0.3150
Epoch 10/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.8863 - lo
ss: 0.3004
Epoch 11/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.8909 - lo
ss: 0.2943
Epoch 12/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8953 - lo
ss: 0.2843
Epoch 13/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8979 - lo
ss: 0.2765
Epoch 14/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8997 - lo
ss: 0.2728
Epoch 15/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9052 - lo
ss: 0.2631
Epoch 16/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9057 - lo
ss: 0.2643
Epoch 17/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9081 - lo
ss: 0.2562
Epoch 18/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9103 - lo
ss: 0.2456
Epoch 19/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9129 - lo
ss: 0.2440
Epoch 20/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9118 - lo
ss: 0.2449
Epoch 21/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9155 - lo
ss: 0.2390
Epoch 22/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9163 - lo
ss: 0.2337
Epoch 23/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9196 - lo
```

```
ss: 0.2273
Epoch 24/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9172 - lo
ss: 0.2332
Epoch 25/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9190 - lo
ss: 0.2300
Epoch 26/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9221 - lo
ss: 0.2236
Epoch 27/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9216 - lo
ss: 0.2235
Epoch 28/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9233 - lo
ss: 0.2215
Epoch 29/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9233 - lo
ss: 0.2191
Epoch 30/30
1447/1447 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9251 - lo
ss: 0.2148
```

Out[192]:
```
<keras.src.callbacks.history.History at 0x19e03497f70>
```

In [193]:
```
# Testing the data
model.evaluate(X_test, y_test)
```

```
582/582 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8443 - los
s: 0.5030
```

Out[193]:
```
[0.5132427215576172, 0.8417571783065796]
```

In [194]:
```
# Saving the model
model.save('Loan_Approval.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.
save()` or `keras.saving.save_model(model)`. This file format is c
onsidered legacy. We recommend using instead the native Keras form
at, e.g. `model.save('my_model.keras')` or `keras.saving.save_mode
l(model, 'my_model.keras')`.
```

In [195]:
```
# Loading the trained model
from keras.models import load_model
model_load = load_model('Loan_Approval.h5')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics h
ave yet to be built. `model.compile_metrics` will be empty until y
ou train or evaluate the model.
```

In [199]:
```
# Predicting the values on Test data set
y_pred = (model_load.predict(X_test)>0.5).astype('int32')
```

```
582/582 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

In [204]:
```
y_pred
```

Out[204]:
```
array([[1],
       [1],
       [0],
       ...,
       [1],
       [0],
       [1]])
```

In [201]:
```
# Testing the model on New Input Data
import joblib

new_data = {'Income':500000,'Age':30, 'Experience':5,'Married/Single

# Converting to pandas DataFrame
df_1 = pd.DataFrame([new_data])

# Concatenation of the new data to the original dataset
new_df = pd.concat([X_old, df_1], ignore_index=True)

# Converting the categorical variables to dummy/indicator variables
new_df = pd.get_dummies(new_df, drop_first=True)

# Extracting the last row(new data) for prediction
new_data_1=new_df.tail(1)
new_data_1

# Loading the scaler
scaling = joblib.load('feature_scaling.pkl')
new_data_1_scaled = scaling.transform(new_data_1)

# Loading the model
model_load = load_model('Loan_Approval.h5')

# Making a prediction
output = (model_load.predict(new_data_1_scaled)>0.5).astype('int32')

# Interpreting the final result
if output == 1:
    print("The Client is High Risk")
else:
    print("The Client is Low Risk")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics h
ave yet to be built. `model.compile_metrics` will be empty until y
ou train or evaluate the model.

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
The Client is High Risk
```

In [205]:
```
from sklearn.metrics import accuracy_score, classification_report, c

print(classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support

           0       0.81      0.86      0.84      8747
           1       0.87      0.83      0.85      9851

    accuracy                           0.84     18598
   macro avg       0.84      0.84      0.84     18598
weighted avg       0.84      0.84      0.84     18598
```

In [ ]: