loan-default-prediction-report-3

June 17, 2024

1 Loan Default Prediction Report

2 Prasad Khandake

3 khandkep02@gmail.com

4 Predict if the client is high risk or low risk if we were to provide them loan using machine learning

Introduction

Creating a predictive model for loan default starts with gathering historical loan data. This data undergoes preprocessing steps such as handling missing values, encoding categorical variables, and selecting relevant features like credit scores and employment history. Machine learning algorithms, including XGBoost, are then trained on this prepared data to predict the risk of default. The performance of these models is assessed using metrics such as accuracy and precision. The resulting predictions help in evaluating risk and making informed decisions, such as modifying loan terms or declining high-risk applications. Python's machine learning libraries play a crucial role in developing effective models for risk assessment and management in lending.

Predicting Loan Default

Learning Outcomes

1. Understanding Loan Default Prediction: Gain insights into the significance of loan default prediction in financial risk assessment and decision-making.

2. Data Preprocessing Techniques: Learn essential data preprocessing steps, including handling missing values, encoding categorical variables, and feature selection.

3. Application of Machine Learning Algorithms: Understand how to apply machine learning algorithms like XGBoost and Random Forest for predicting loan defaults using Python.

4. Performance Evaluation Metrics: Learn to evaluate model performance using metrics such as accuracy, precision, recall, F1-score, and AUC in binary classification tasks.

Types of Default

Default can occur in various forms, impacting both secured and unsecured debts. A secured debt default happens when a borrower fails to make payments on loans backed by collateral, such as a mortgage loan secured by property or a business loan secured by assets. Corporate bond default occurs when a company cannot meet its coupon payments. Unsecured debt defaults, such as credit card debt, also impact the borrower's credit and future borrowing capacity. Understanding these scenarios is crucial in financial modeling and evaluating the effectiveness of different learning methods, including linear regression and deep learning algorithms.

Why Do People Borrow, and Why Do Lenders Exist?

Debt is a critical resource for individuals and businesses, allowing them to make significant investments in homes, vehicles, and other major expenses. While loans can provide financial opportunities, they also come with substantial risks. Lending is essential for economic growth, supporting both individuals and enterprises globally. As economies become more interconnected, the demand for capital has increased, leading to a rise in retail, SME, and commercial borrowers. This surge has boosted revenues for many financial institutions but has also presented new challenges.

Recently, there has been a noticeable increase in loan defaults, affecting the profitability and risk management strategies of lenders. This trend highlights the need for effective loan management, supported by advanced techniques such as support vector machines and gradient-based models. These sophisticated methods help accurately assess loan amounts, repayment probabilities, and overall risk profiles.

About the Data

For organizations aiming to predict defaults on consumer lending products, leveraging historical client behavior data is crucial. Analyzing past patterns enables them to identify high-risk and low-risk consumers, optimizing lending decisions for future clients.

By utilizing advanced techniques such as boosting, organizations can enhance the predictive power of their models. These techniques help in identifying subtle patterns and signals indicative of default risk, allowing the development of robust predictive models tailored to specific lending contexts.

Thorough validation processes ensure the reliability and accuracy of these models, validating their performance on diverse datasets and ensuring their effectiveness in real-world scenarios. Continuous refinement and validation of predictive models enable organizations to make informed lending decisions, mitigating risks and maximizing returns.

In rapidly evolving lending landscapes, such as in China, predictive analytics capabilities are particularly valuable. With the increasing complexity of consumer behavior and financial transactions, leveraging data-driven insights is indispensable for effective risk management and decision-making in the lending sector.

The data includes demographic features of each customer and a target variable indicating whether they will default on the loan.

First, we import the libraries and load the dataset.

[105]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[106]:
```
import numpy as np # linear algebra
import pandas as pd # data processing
```

[107]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_theme(style = "darkgrid")
```

Now, we read the data.

[108]:
```
import json

# Use 'rb' mode to read the file as binary, which can handle potential encoding␣
 ↪issues
with open('/content/drive/My Drive/Dataset_files/loan_approval_dataset.json',␣
 ↪'rb') as f:
    # Decode the binary data assuming UTF-8 encoding
    data = json.loads(f.read().decode('utf-8'))

# Now you can work with the 'data' variable as a Python dictionary or list
print(data)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

[109]:
```
# Load the data
df = pd.DataFrame(data)
```

4.1 1. Data Exploration

Check Data Types and Missing Values

Understanding the Dataset

First, we start with understanding the data set and how is the data distributed.

[110]:
```
rows, columns = df.shape
print('Rows:', rows)
print('Columns:', columns)
```

```
Rows: 252000
Columns: 13
```

So, we see that the data is 252000 rows, that is 252000 data points and 13 columns, that is 13 features. Out of13 features, 12 are input features and 1 is output feature.

Now we check the data types and other information.

[111]:
```
df.head()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 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
```

Understanding Data Types

Upon examination, it's apparent that half of the features are numeric, while the other half are strings, indicating they are likely categorical features.

Numeric Data:

Numeric data represents measurable quantities of a phenomenon. In data science, it is referred to as "quantitative data" because it quantifies the amount or extent of the object it represents.

Categorical Data:

Categorical data describes the qualities or properties of a phenomenon that can be named or categorized. It is referred to as "qualitative data" in data science, as it characterizes the quality or type of the entity it represents.

Let us check if there are any missing values in the data.

[112]:
```
df.isnull().sum()
```

[112]:
```
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
```

So, there is no missing or empty data here.

Let us check the data column names.

[113]:
```
df.columns
```

[113]:
```
Index(['Id', 'Income', 'Age', 'Experience', 'Married/Single',
       'House_Ownership', 'Car_Ownership', 'Profession', 'CITY', 'STATE',
       'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS', 'Risk_Flag'],
      dtype='object')
```

So, we get the names of the data features.

Analyzing Numerical Columns

First, we start with the analysis of numerical data.

[114]:
```
df.describe()
```

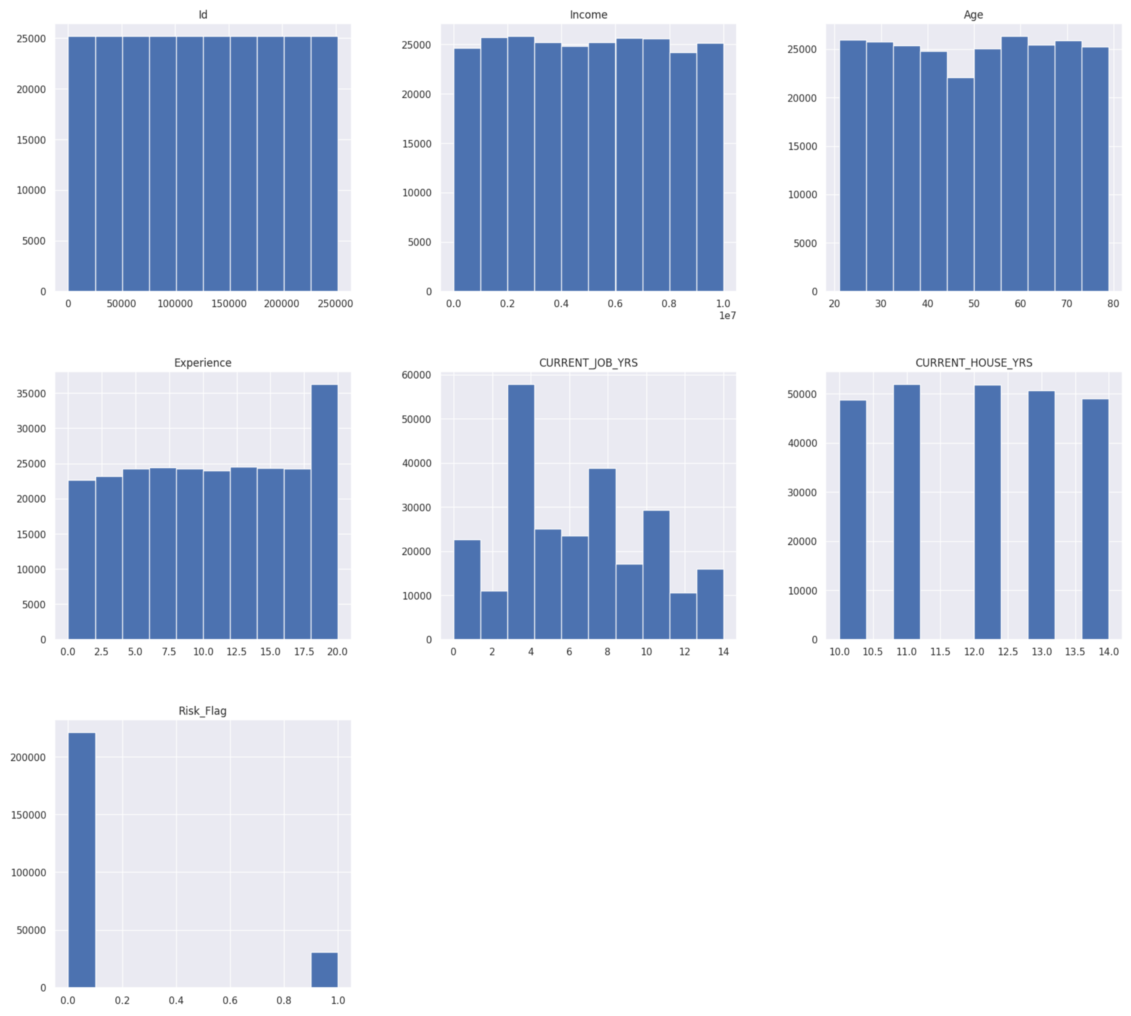
[114]:
```
                  Id        Income            Age     Experience  \
count  252000.000000  2.520000e+05  252000.000000  252000.000000
mean   126000.500000  4.997117e+06      49.954071      10.084437
std     72746.278255  2.878311e+06      17.063855       6.002590
min         1.000000  1.031000e+04      21.000000       0.000000
25%     63000.750000  2.503015e+06      35.000000       5.000000
50%    126000.500000  5.000694e+06      50.000000      10.000000
75%    189000.250000  7.477502e+06      65.000000      15.000000
max    252000.000000  9.999938e+06      79.000000      20.000000

       CURRENT_JOB_YRS  CURRENT_HOUSE_YRS      Risk_Flag
count    252000.000000      252000.000000  252000.000000
mean          6.333877          11.997794       0.123000
std           3.647053           1.399037       0.328438
min           0.000000          10.000000       0.000000
25%           3.000000          11.000000       0.000000
50%           6.000000          12.000000       0.000000
75%           9.000000          13.000000       0.000000
```

```
max           14.000000           14.000000        1.000000
```

Now, we check the data distribution.

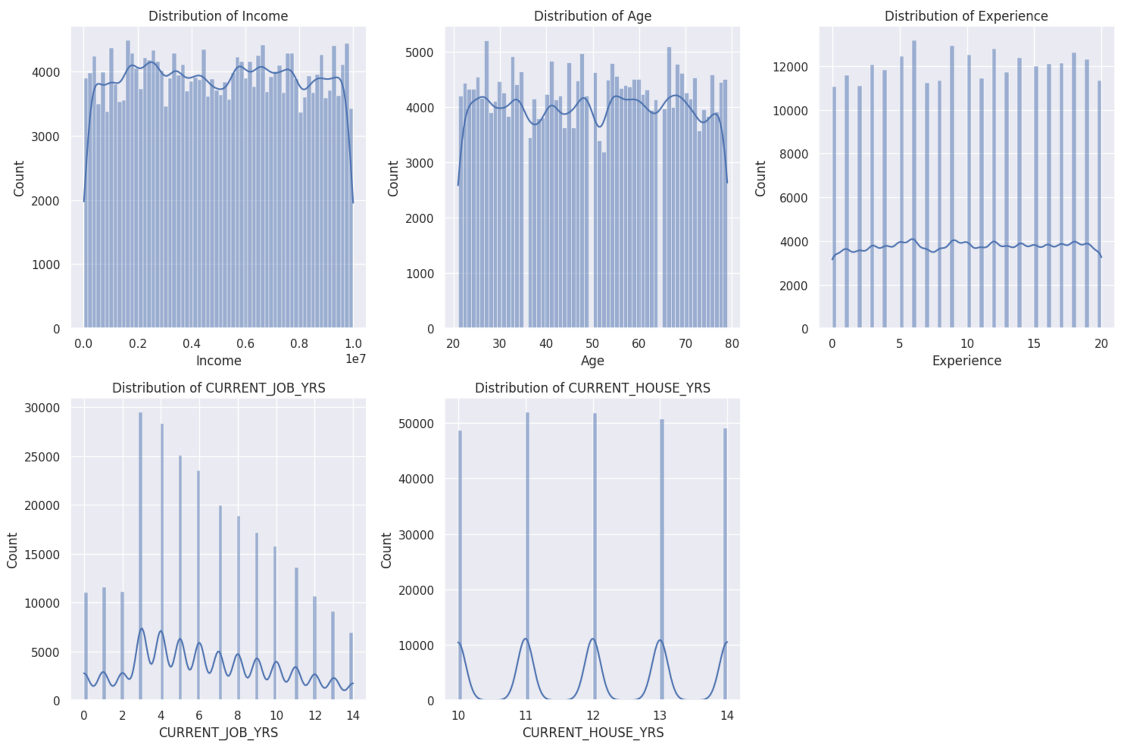
```
[115]: df.hist( figsize = (22, 20) )
       plt.show()
```



Distribution of Numerical Features

```
[116]: numerical_cols = ['Income', 'Age', 'Experience', 'CURRENT_JOB_YRS',␣
         ↪'CURRENT_HOUSE_YRS']

       plt.figure(figsize=(15, 10))
       for i, col in enumerate(numerical_cols, 1):
           plt.subplot(2, 3, i)
           sns.histplot(df[col], kde=True)
           plt.title(f'Distribution of {col}')
       plt.tight_layout()
       plt.show()
```



Now, we check the count of the target variable.

[117]:
```
df["Risk_Flag"].value_counts()
```

[117]:
```
Risk_Flag
0    221004
1     30996
Name: count, dtype: int64
```

Only a small part of the target variable consists of people who default on loans.

4.2 2. Advanced Feature Selection

Correlation Matrix

Plot the correlation matrix to understand the relationships between features.

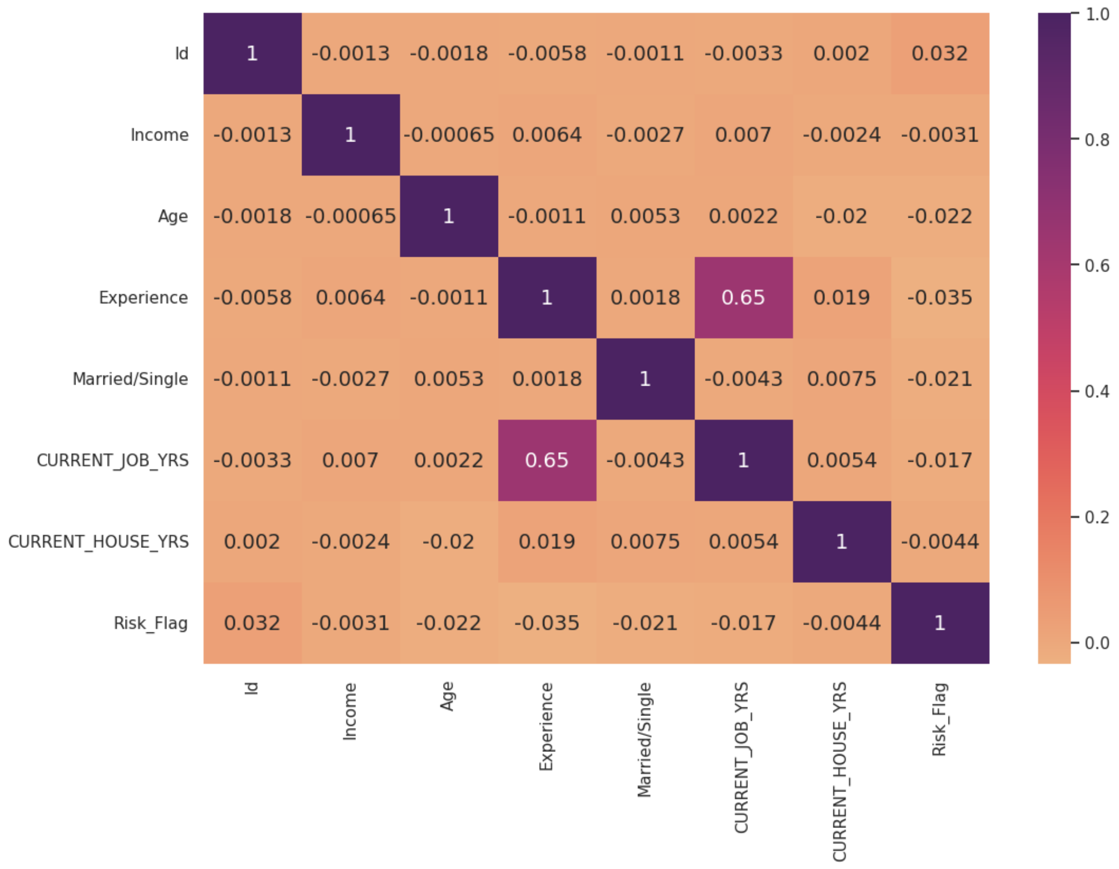
Now, we plot the correlation plot.

[118]:
```
# Convert non-numeric columns to numeric if applicable.

df['Married/Single'] = df['Married/Single'].replace({'single': 0, 'married': 1})
```

```
# Or drop non-numeric columns if they are not relevant for correlation:
df_numeric = df.select_dtypes(include=['number'])

# Calculate correlation on the numeric DataFrame
fig, ax = plt.subplots(figsize=(12, 8))
corr_matrix = df_numeric.corr()
corr_heatmap = sns.heatmap(corr_matrix, cmap="flare", annot=True, ax=ax,␣
↪annot_kws={"size": 14})
plt.show()
```



Analyzing Categorical Features

Now, we proceed with the analysis of categorical features.

First, we define a function to create the plots.

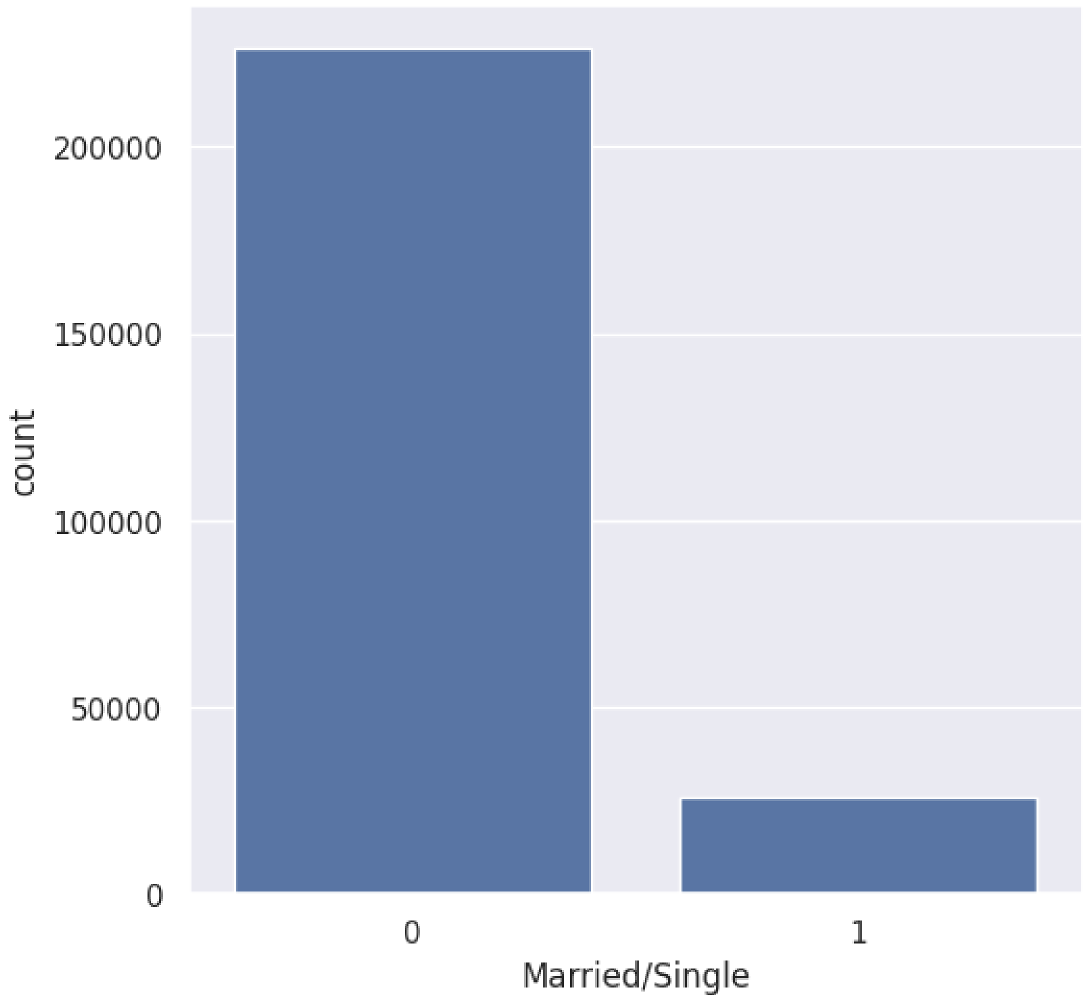
[119]:
```
def categorical_valcount_hist(feature):
    print(df[feature].value_counts())
    fig, ax = plt.subplots( figsize = (6,6) )
    sns.countplot(x=feature, ax=ax, data=df)
    plt.show()
```

4.3 3. Data Visualization

First, we check the count of married people vs single people.

[120]:
```
categorical_valcount_hist("Married/Single")
```

```
Married/Single
0    226272
1     25728
Name: count, dtype: int64
```

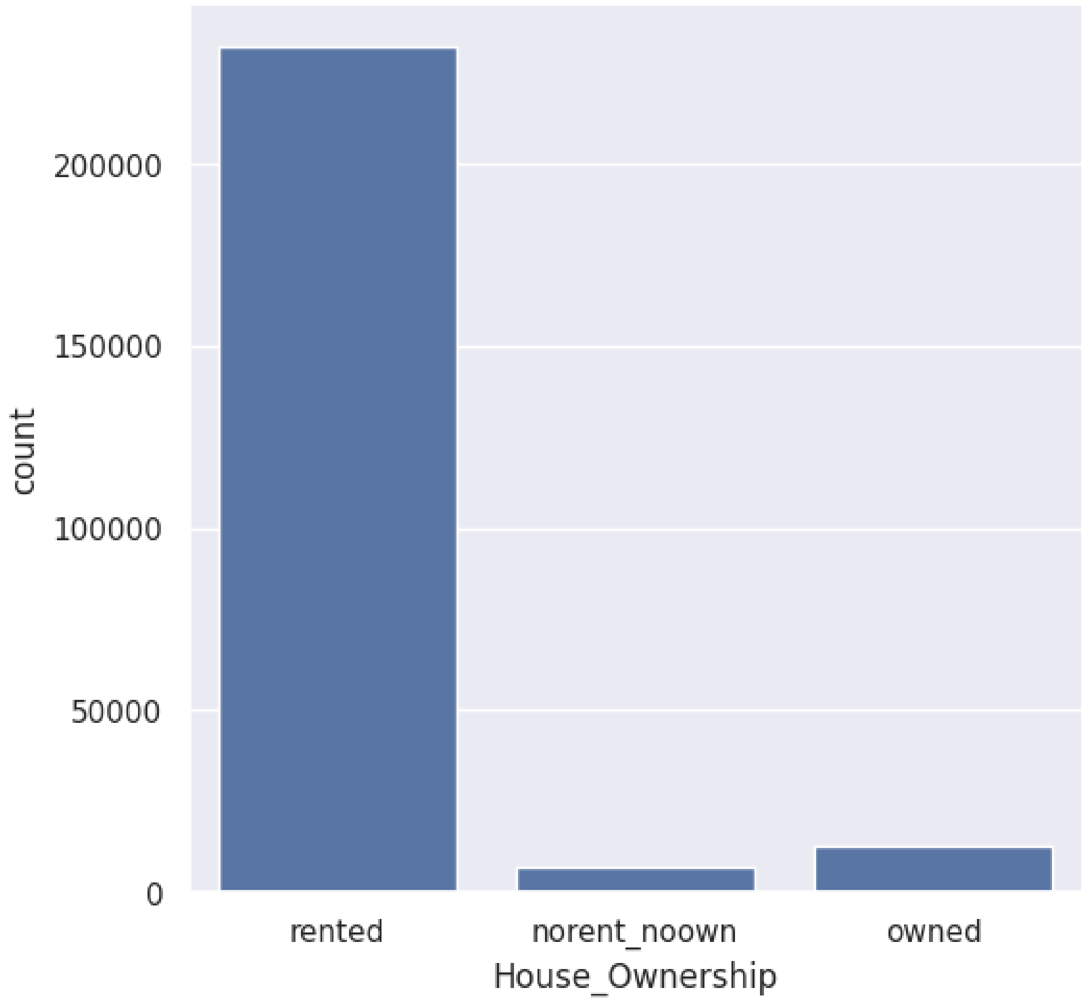


So, the majority of the people are single.

Now, we check the count of house ownership.

```
[121]: categorical_valcount_hist("House_Ownership")
```

```
House_Ownership
rented          231898
owned            12918
norent_noown      7184
Name: count, dtype: int64
```



Now, let us check the count of states.

```
[122]: print( "Total categories in STATE:", len( df["STATE"].unique() ) )
       print()
       print( df["STATE"].value_counts() )
```

```
Total categories in STATE: 29
```

```
STATE
Uttar_Pradesh        28400
Maharashtra          25562
Andhra_Pradesh       25297
West_Bengal          23483
Bihar                19780
Tamil_Nadu           16537
Madhya_Pradesh       14122
Karnataka            11855
Gujarat              11408
Rajasthan             9174
Jharkhand             8965
Haryana               7890
Telangana             7524
Assam                 7062
Kerala                5805
Delhi                 5490
Punjab                4720
Odisha                4658
Chhattisgarh          3834
Uttarakhand           1874
Jammu_and_Kashmir     1780
Puducherry            1433
Mizoram                849
Manipur                849
Himachal_Pradesh       833
Tripura                809
Uttar_Pradesh[5]       743
Chandigarh             656
Sikkim                 608
Name: count, dtype: int64
```

Now, we check the count of professions.

```
[123]: print( "Total categories in Profession:", len( df["Profession"].unique() ) )
       print()
       df["Profession"].value_counts()
```

```
Total categories in Profession: 51
```

```
[123]: Profession
       Physician                     5957
       Statistician                  5806
       Web_designer                  5397
       Psychologist                  5390
       Computer_hardware_engineer    5372
       Drafter                       5359
```

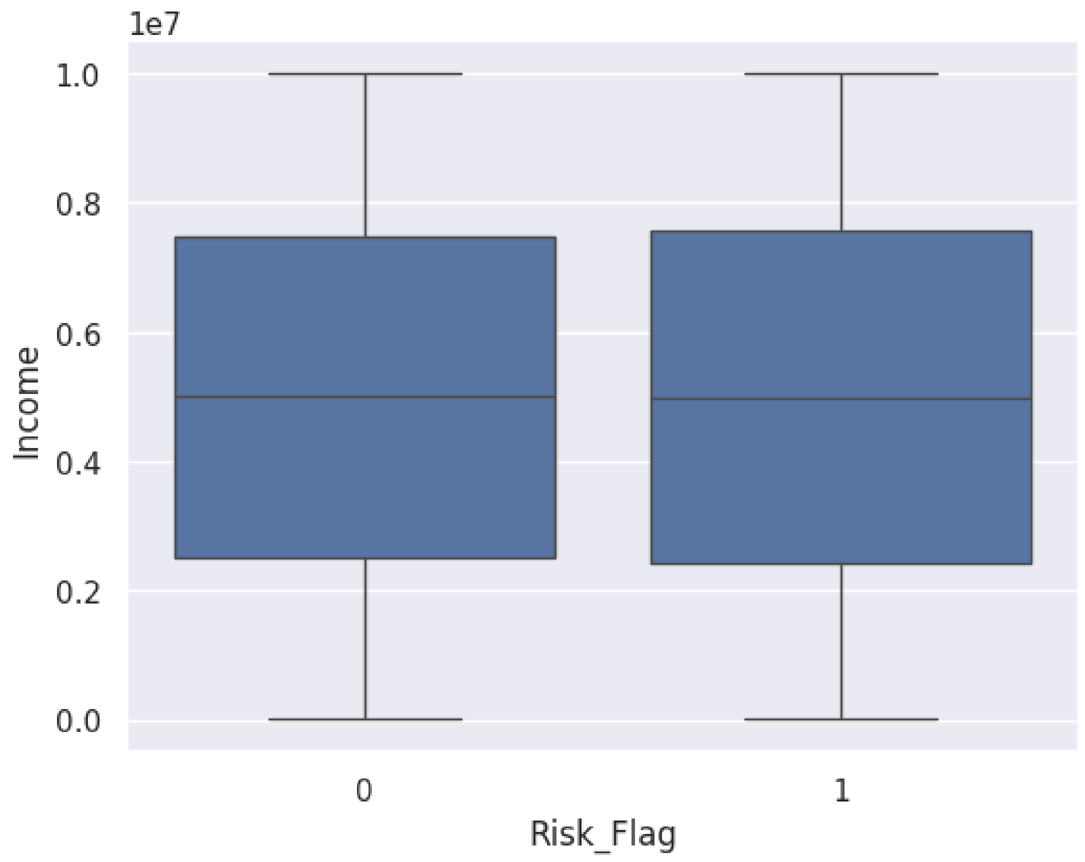
```
Magistrate                 5357
Fashion_Designer           5304
Air_traffic_controller     5281
Comedian                   5259
Industrial_Engineer        5250
Mechanical_engineer        5217
Chemical_engineer          5205
Technical_writer           5195
Hotel_Manager              5178
Financial_Analyst          5167
Graphic_Designer           5166
Flight_attendant           5128
Biomedical_Engineer        5127
Secretary                  5061
Software_Developer         5053
Petroleum_Engineer         5041
Police_officer             5035
Computer_operator          4990
Politician                 4944
Microbiologist             4881
Technician                 4864
Artist                     4861
Lawyer                     4818
Consultant                 4808
Dentist                    4782
Scientist                  4781
Surgeon                    4772
Aviator                    4758
Technology_specialist      4737
Design_Engineer            4729
Surveyor                   4714
Geologist                  4672
Analyst                    4668
Army_officer               4661
Architect                  4657
Chef                       4635
Librarian                  4628
Civil_engineer             4616
Designer                   4598
Economist                  4573
Firefighter                4507
Chartered_Accountant       4493
Civil_servant              4413
Official                   4087
Engineer                   4048
Name: count, dtype: int64
```

Data Analysis

Now, we start with understanding the relationship between the different data features.

```
[124]: sns.boxplot(x ="Risk_Flag",y="Income" ,data = df)
```

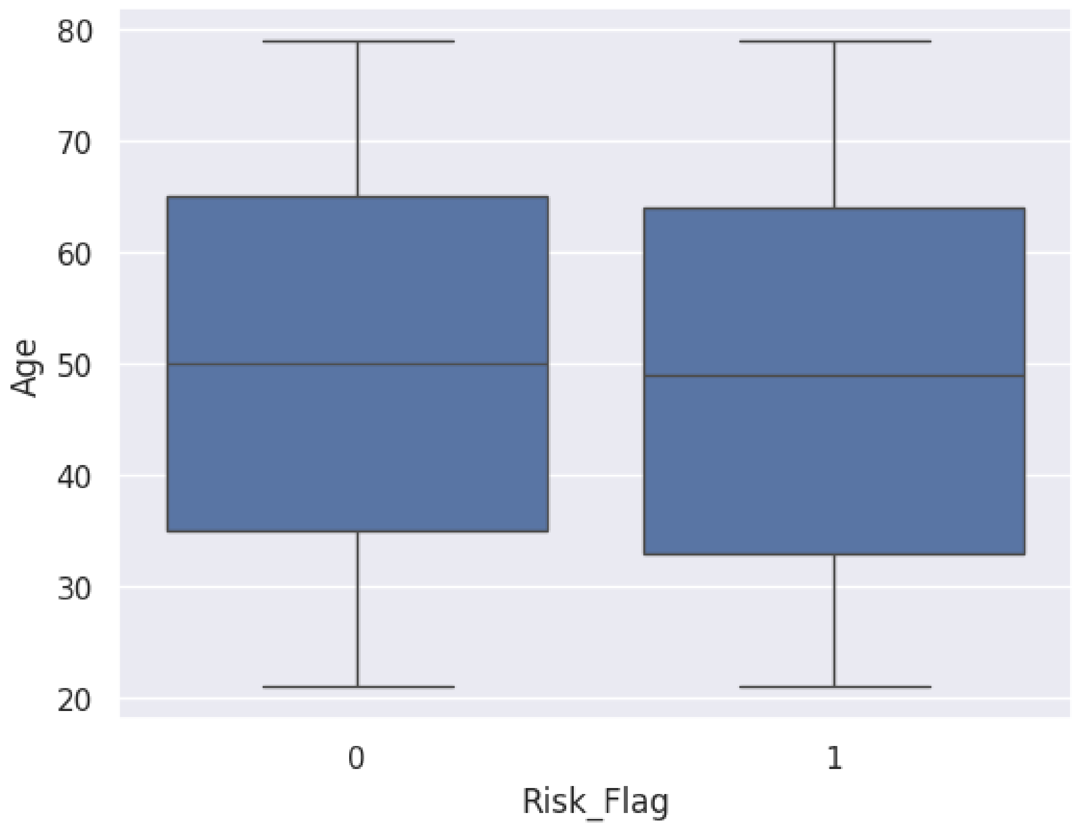
```
[124]: <Axes: xlabel='Risk_Flag', ylabel='Income'>
```



Now, we see the relationship between the flag variable and age.

```
[125]: sns.boxplot(x ="Risk_Flag",y="Age" ,data = df)
```

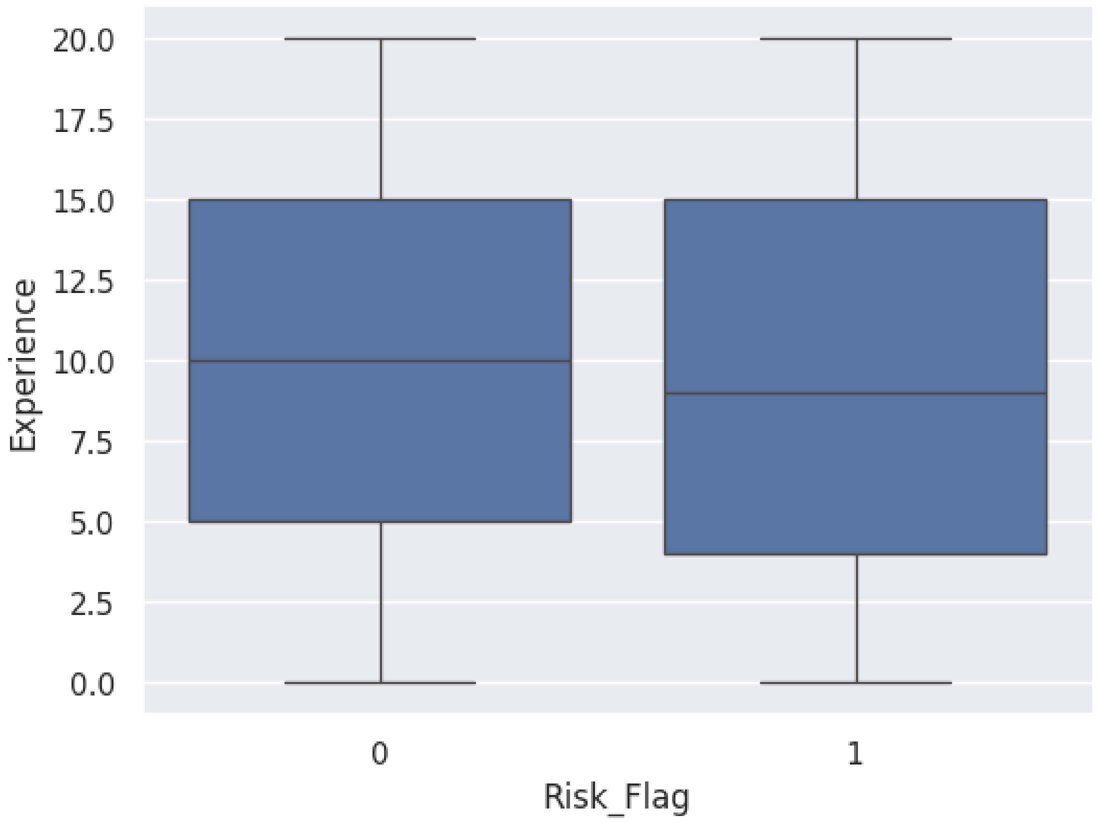
```
[125]: <Axes: xlabel='Risk_Flag', ylabel='Age'>
```



we see the relationship between the flag variable and Experience.

[126]:
```
sns.boxplot(x ="Risk_Flag",y="Experience" ,data = df)
```

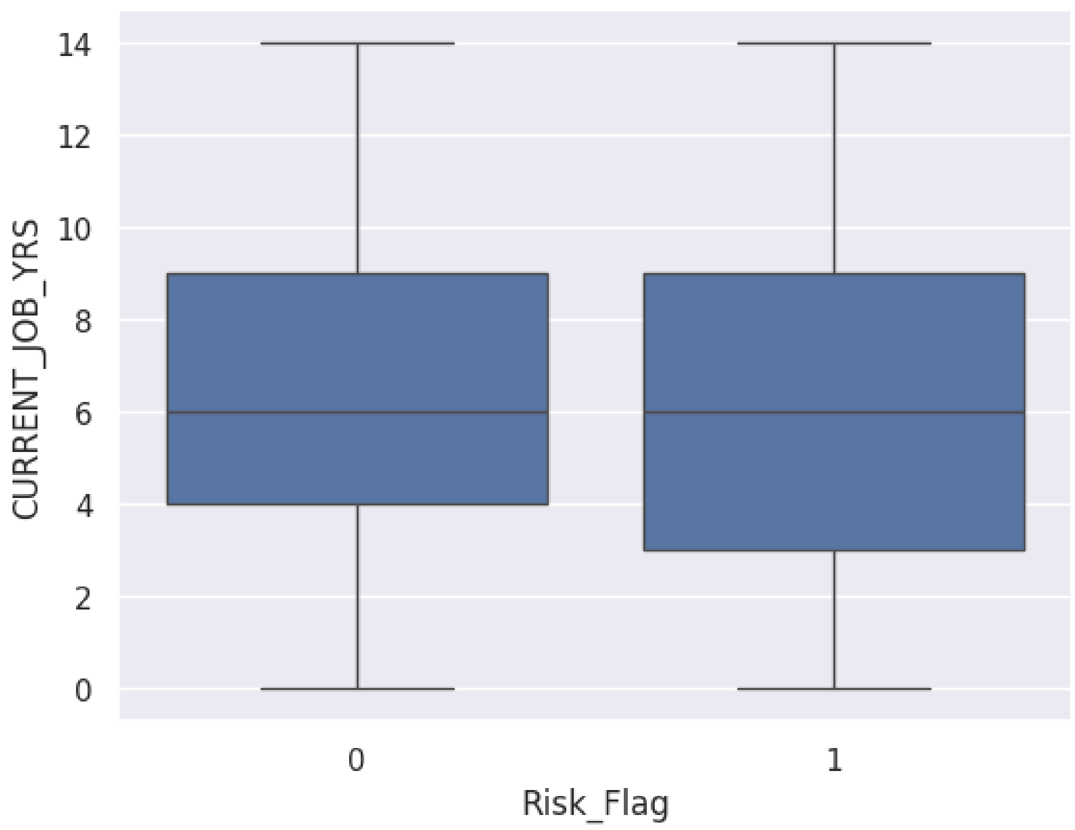
[126]:
```
<Axes: xlabel='Risk_Flag', ylabel='Experience'>
```



we see the relationship between the flag variable and CURRENT_JOB_YRS.

```
[127]: sns.boxplot(x ="Risk_Flag",y="CURRENT_JOB_YRS" ,data = df)
```

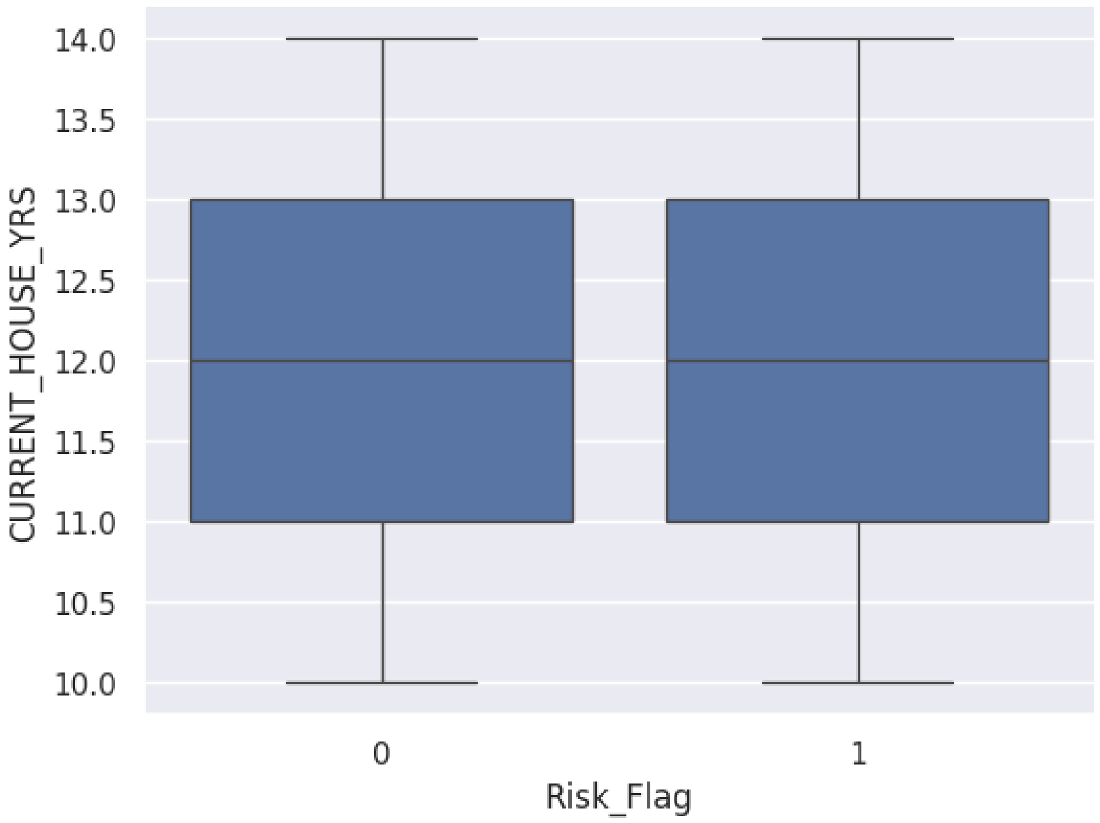
```
[127]: <Axes: xlabel='Risk_Flag', ylabel='CURRENT_JOB_YRS'>
```



we see the relationship between the flag variable and CURRENT_HOUSE_YRS.

```
[128]: sns.boxplot(x ="Risk_Flag",y="CURRENT_HOUSE_YRS" ,data = df)
```

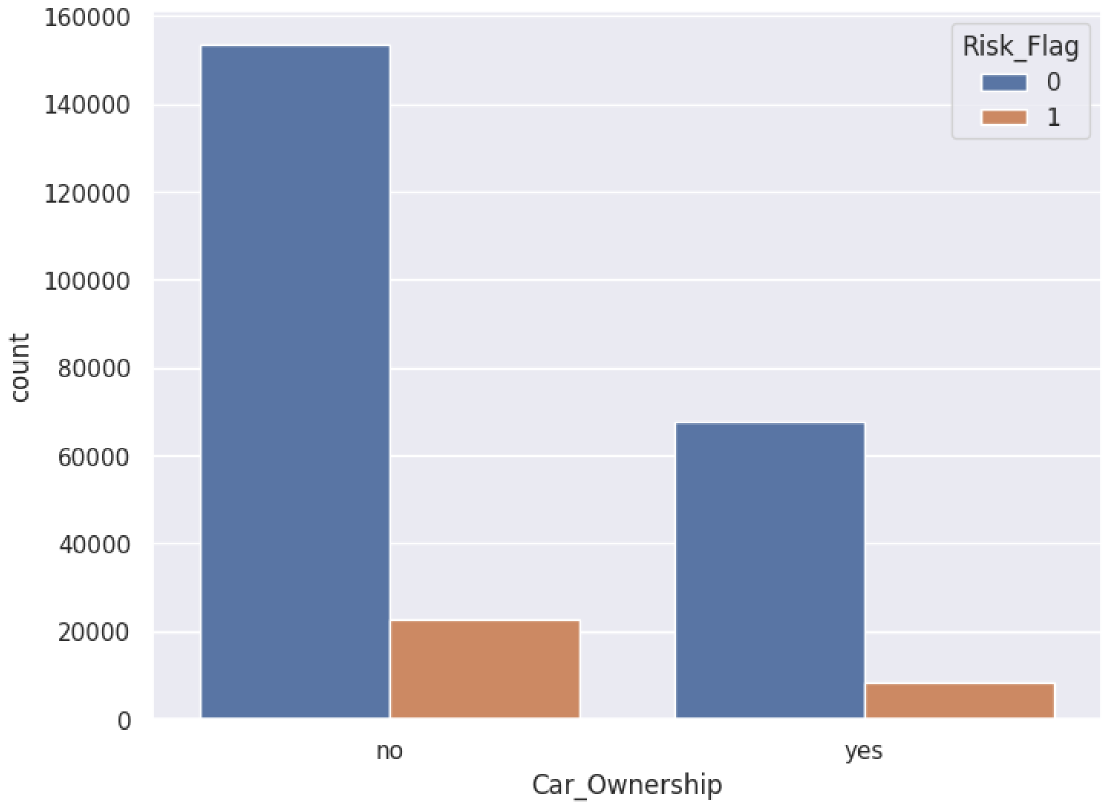
```
[128]: <Axes: xlabel='Risk_Flag', ylabel='CURRENT_HOUSE_YRS'>
```



we see the relationship between the flag variable and Car_Ownership.

```
[129]: fig, ax = plt.subplots( figsize = (8,6) )
       sns.countplot(x='Car_Ownership', hue='Risk_Flag', ax=ax, data=df)
```

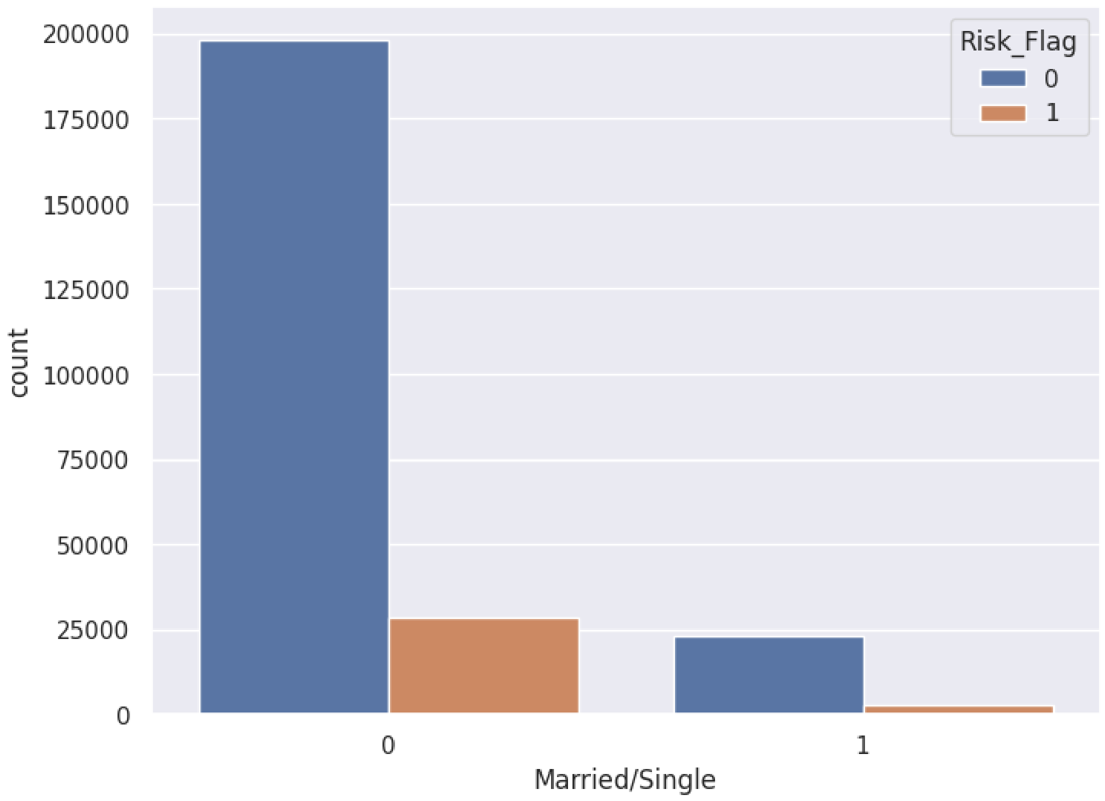
```
[129]: <Axes: xlabel='Car_Ownership', ylabel='count'>
```



we see the relationship between the flag variable and Married/Single.

```
[130]: fig, ax = plt.subplots( figsize = (8,6) )
       sns.countplot( x='Married/Single', hue='Risk_Flag', data=df)
```

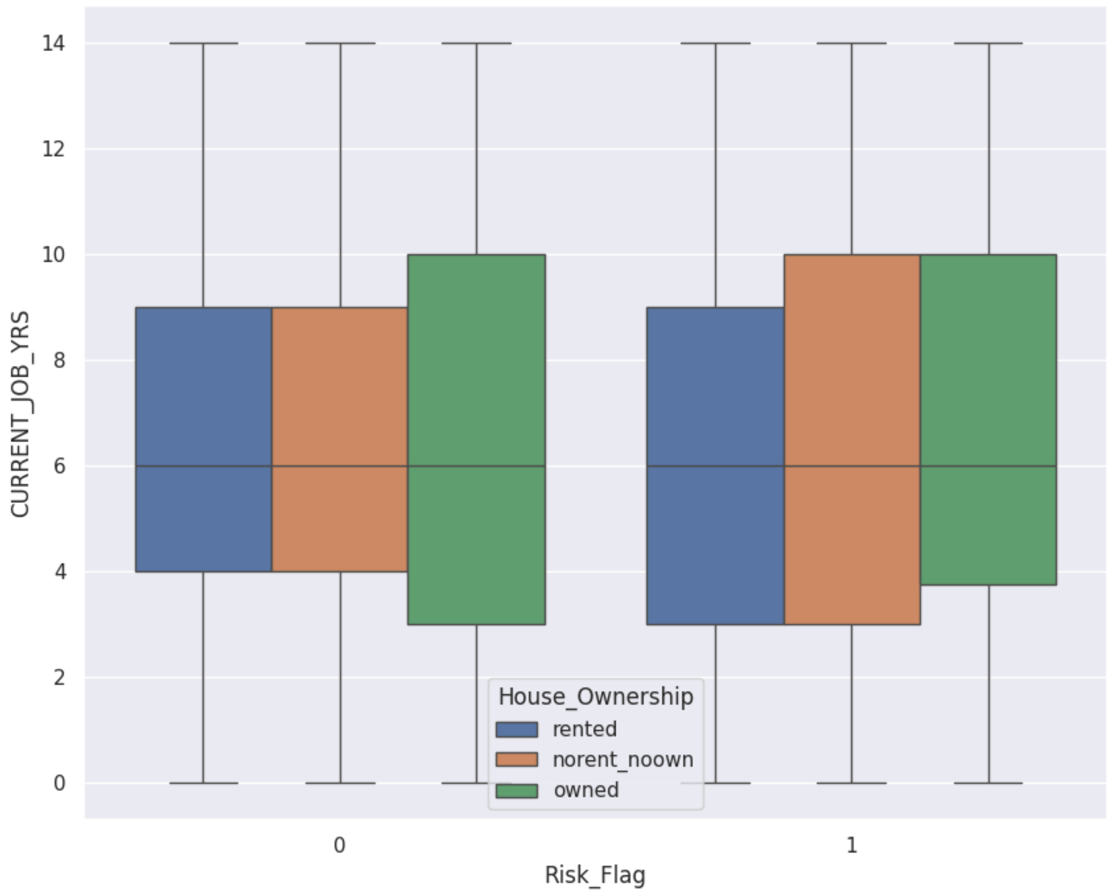
```
[130]: <Axes: xlabel='Married/Single', ylabel='count'>
```



Now, we see the relationship between the flag variable and CURRENT_JOB_YRS, House_Ownership.

```
[131]: fig, ax = plt.subplots( figsize = (10,8) )
       sns.boxplot(x = "Risk_Flag", y = "CURRENT_JOB_YRS", hue='House_Ownership', data␣
        ↪= df)
```

```
[131]: <Axes: xlabel='Risk_Flag', ylabel='CURRENT_JOB_YRS'>
```



4.4 4. Feature Engineering

Convert Categorical Features to Numerical using Label Encoding

Encoding

Data preparation is an essential step in data science before moving on to modeling. During this process, several tasks must be completed, one of which is encoding categorical data. In real-world datasets, categorical string values are common, but most machine learning models can only process integer values or other numerical formats. This is because all models perform mathematical operations using various tools and methodologies.

Encoding categorical data involves transforming these string values into an integer format. This transformation allows the encoded data to be fed into models, thereby improving prediction accuracy.

We will apply encoding to the categorical features.

[132]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import category_encoders as ce
```

```
label_encoder = LabelEncoder()
for col in ['Married/Single','Car_Ownership']:
  df[col] = label_encoder.fit_transform(df[col])
onehot_encoder = OneHotEncoder(sparse = False)
df['House_Ownership'] = onehot_encoder.fit_transform(df['House_Ownership'].
 ↪values.reshape(-1, 1) )
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868:
FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will
be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its
default value.
  warnings.warn(
```

[133]:
```
high_card_features = ['Profession', 'CITY', 'STATE']
count_encoder = ce.CountEncoder()
# Transform the features, rename the columns with the _count suffix, and join␣
 ↪to dataframe
count_encoded = count_encoder.fit_transform( df[high_card_features] )
df = df.join(count_encoded.add_suffix("_count"))
```

[134]:
```
df= df.drop(labels=['Profession', 'CITY', 'STATE'], axis=1)
```

After the feature engineering part is complete, we shall split the data into training and testing sets.

4.5 5. Split the Data into Training and Testing Sets

Prepare the Features and Target Variable

The train-test split is a fundamental technique for measuring the performance of machine learning models, particularly in prediction-based applications. This straightforward procedure allows us to compare our model's outcomes with actual results. Typically, the data is divided such that 70% is used for training and 30% for testing.

To evaluate the effectiveness of our machine learning model, the dataset is split into two sets: the training set and the test set. The training set, comprising 70% of the data, is used to train the model, allowing it to learn from known statistics. The test set, making up the remaining 30%, is used exclusively for making predictions and assessing the model's performance.

It is an important part of the ML chain.

[135]:
```
x = df.drop("Risk_Flag", axis=1)
y = df["Risk_Flag"]
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,␣
 ↪stratify = y, random_state = 7)
```

We have taken the test size to be 20% of the entire data.

4.6 6. Train a Random Forest Classifier Model

Train and Evaluate the Model

Random Forest Classifier

Tree-based algorithms, such as random forests, are vital for loan default prediction and credit risk assessment. These algorithms excel in handling both classification and regression tasks, making them highly valuable in analyzing loan applications. By generating predictions based on training samples, they offer high accuracy and stability, essential for identifying potential defaulters.

In loan default prediction, tree-based algorithms help minimize false negatives and false positives, ensuring robust risk assessment. While individual decision trees may overfit the training data, random forests mitigate this issue by averaging predictions from multiple trees, leading to improved prediction accuracy.

Academic research often explores the efficacy of tree-based algorithms in loan default prediction, with studies published in reputable journals. Authors typically provide DOIs for their work, facilitating citation and further research. Additionally, comparing tree-based models with logistic regression models can offer insights into the strengths and limitations of each approach in credit risk assessment.

Now, we train the model and perform the predictions.

[136]:
```python
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
rf_clf = RandomForestClassifier(criterion='gini', bootstrap=True,␣
 ↪random_state=100)
smote_sampler = SMOTE(random_state=9)
pipeline = Pipeline(steps = [['smote', smote_sampler],
                             ['classifier', rf_clf]])
pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)
```

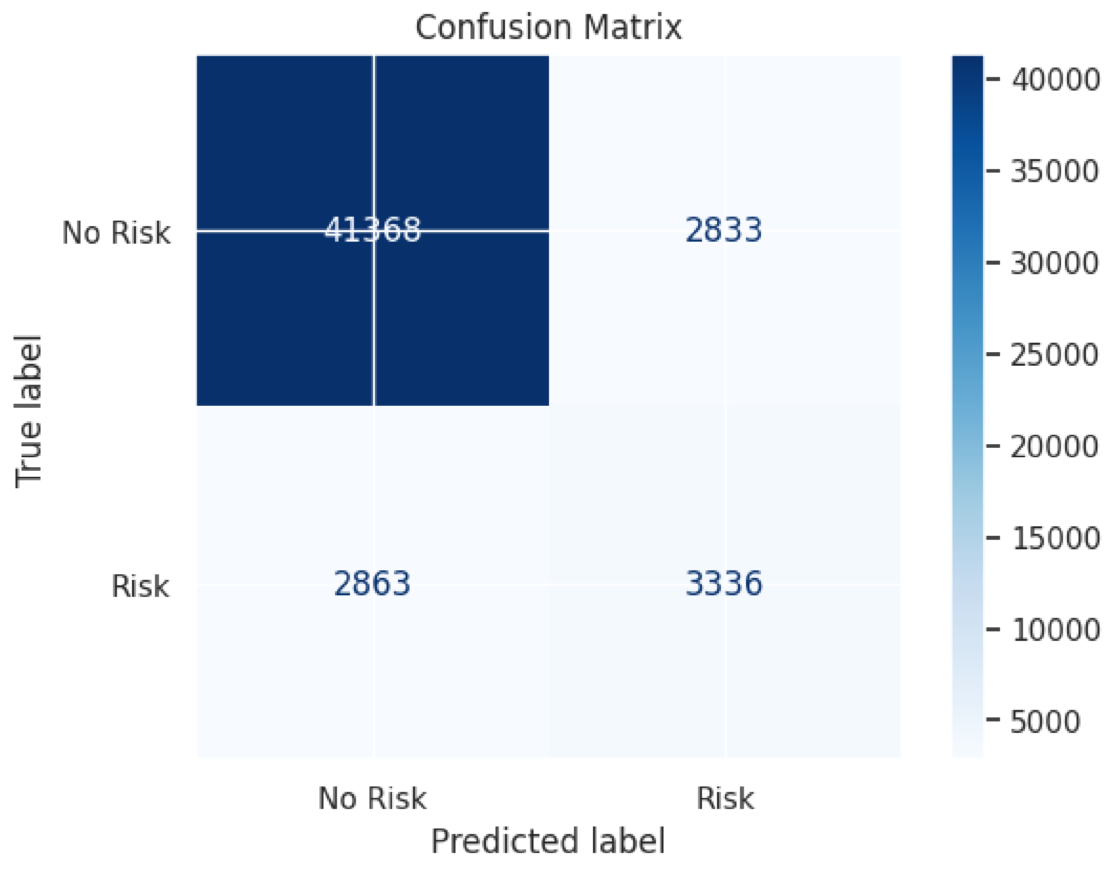
4.7 7. Model Evaluation

Confusion Matrix

Plot the confusion matrix to evaluate the model's performance.

[138]:
```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

conf_matrix = confusion_matrix(y_test, y_pred)
cmd = ConfusionMatrixDisplay(conf_matrix, display_labels=['No Risk', 'Risk'])
cmd.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```



Now, we check the accuracy scores.

```
[137]: from sklearn.metrics import confusion_matrix, precision_score, recall_score,␣
       ↪f1_score, accuracy_score, roc_auc_score
       print("--------------------------TEST SCORES-----------------------")
       print(f"Recall: { round(recall_score(y_test, y_pred)*100, 4) }")
       print(f"Precision: { round(precision_score(y_test, y_pred)*100, 4) }")
       print(f"F1-Score: { round(f1_score(y_test, y_pred)*100, 4) }")
       print(f"Accuracy score: { round(accuracy_score(y_test, y_pred)*100, 4) }")
       print(f"AUC Score: { round(roc_auc_score(y_test, y_pred)*100, 4) }")
```

```
--------------------------TEST SCORES-----------------------
Recall: 53.8151
Precision: 54.0768
F1-Score: 53.9457
Accuracy score: 88.6984
AUC Score: 73.7029
```

The accuracy scores might not be up to the mark, but this is the overall process of predicting loan default.

4.8 8. Conclusion

Predicting loan default involves thorough exploratory data analysis (EDA) to understand dataset characteristics. Utilizing Python libraries and techniques like boosting, random forest classifiers, and logistic regression, classification models are developed using artificial intelligence algorithms.

Evaluation metrics such as accuracy, precision, recall, F1-score, and AUC assess model performance in binary classification tasks. International conferences on data science and AI foster collaboration and innovation in risk assessment and management. This integrated approach enables effective prediction of loan defaults, which is crucial for managing risk in the financial sector.

Key Takeaways

Random Forest Approach:

Suitable for classification and regression tasks on large datasets with many entries and features, even when missing values are present. It offers high accuracy while avoiding overfitting.

Feature Significance:

Random forests provide relative feature importance, helping to identify the most crucial features. This method is more interpretable than neural networks but less so than decision trees.

Categorical Feature Encoding:

Encoding is necessary for categorical features to be processed by machine learning algorithms.

Demographic Impact:

Loan default prediction is significantly influenced by demographic factors, with lower-income individuals more likely to default.

Random Forest Classifier Success:

The classification task using the Random Forest Classifier was successfully performed, demonstrating the model's effectiveness in predicting loan defaults.

5 Thank You