Kraddle pro notebook

July 20, 2025

0.0.1 Objective:

The objective is to build a predictive model on this data to help the bank decide on whether to approve a loan to a prospective applicant.

Data Dictionary LoanID -Unique identifier for each loan Age -Age of the borrower Income -Annual income LoanAmount -Total loan amount requested CreditScore -Credit score of the borrower MonthsEmployed- Number of months the borrower has been employed NumCreditLines -Number of credit lines the borrower has InterestRate -Annual interest rate (%) LoanTerm- Term of the loan in months DTIRati - Debt-to-Income ratio (lower is better) Education-Education level of the borrower EmploymentType -Type of employment (e.g. Full-time, Unemployed) MaritalStatus-Marital status (Married, Divorced, etc.) HasMortgage- Whether the borrower has an active mortgage HasDependents- Whether the borrower has dependents LoanPurpose- Purpose of the loan (Auto, Business, Other, etc.) HasCoSigner- Whether the borrower has a co-signer Default- Target variable – whether the loan defaulted (1) or not (0)

[1]:
```
### Import necessary libraries
```

[96]:
```
# this will help in making the Python code more structured automatically (good␣
↪coding practice)
import jupyter_black

jupyter_black.load()

# To filter the warnings
import warnings

warnings.filterwarnings("ignore")

# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)

# libaries to help with data visualization
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Library to split data
from sklearn.model_selection import train_test_split

# To normalise continuous variables
from sklearn.preprocessing import StandardScaler, PowerTransformer

# To tune a model
from sklearn.model_selection import RandomizedSearchCV

# To build model for prediction
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant
from sklearn.linear_model import LogisticRegression  # Logistic Regression
from sklearn.model_selection import cross_val_score
import joblib
import pickle
import json


# To get different metric scores
import sklearn.metrics as metrics
from sklearn.metrics import (
    f1_score,
    make_scorer,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    ConfusionMatrixDisplay,
    precision_recall_curve,
    roc_curve,
)
import pandas as pd

pd.set_option("display.max_columns", None)
```

[98]:
```python
# Loading the dataset - sheet_name parameter is used if there are multiple tabs␣
↪in the excel file.
```

[100]:
```python
df = pd.read_csv(
```

```
    "C:/Users/Munashe Muchinako/OneDrive/Desktop/data science/kraddle proj/
 ↪Loan_default.csv"
)
```

[101]:
```
# copy the data into duplicate variable 'data' to avoid making changes to the␣
 ↪original data
raw_data = df.copy()
```

[102]:
```
# show top 5 rows in the data
raw_data.head(5)
```

[102]:
```
      LoanID  Age  Income  LoanAmount  CreditScore  MonthsEmployed  \
0  I38PQUQS96   56   85994       50587          520              80
1  HPSK72WA7R   69   50432      124440          458              15
2  C1OZ6DPJ8Y   46   84208      129188          451              26
3  V2KKSFM3UN   32   31713       44799          743               0
4  EY08JDHTZP   60   20437        9139          633               8

   NumCreditLines  InterestRate  LoanTerm  DTIRatio    Education  \
0               4         15.23        36      0.44   Bachelor's
1               1          4.81        60      0.68     Master's
2               3         21.17        24      0.31     Master's
3               3          7.07        24      0.23  High School
4               4          6.51        48      0.73   Bachelor's

  EmploymentType MaritalStatus HasMortgage HasDependents LoanPurpose  \
0      Full-time      Divorced         Yes           Yes       Other
1      Full-time       Married          No            No       Other
2     Unemployed      Divorced         Yes           Yes        Auto
3      Full-time       Married          No            No    Business
4     Unemployed      Divorced          No           Yes        Auto

  HasCoSigner  Default
0         Yes        0
1         Yes        0
2          No        1
3          No        0
4          No        0
```

[103]:
```
# Display last 3 rows of the data
raw_data.tail(3)
```

[103]:
```
            LoanID  Age  Income  LoanAmount  CreditScore  MonthsEmployed  \
255344  XQK1UUUNGP   56   84820      208294          597              70
255345  JAO28CPL4H   42   85109       60575          809              40
255346  ZTH91CGL0B   62   22418       18481          636             113
```

```
        NumCreditLines  InterestRate  LoanTerm  DTIRatio    Education  \
255344               3          5.29        60      0.50  High School
255345               1         20.90        48      0.44  High School
255346               2          6.73        12      0.48   Bachelor's

       EmploymentType MaritalStatus HasMortgage HasDependents LoanPurpose  \
255344  Self-employed       Married         Yes           Yes        Auto
255345      Part-time        Single         Yes           Yes       Other
255346     Unemployed      Divorced         Yes            No   Education

       HasCoSigner  Default
255344         Yes        0
255345          No        0
255346         Yes        0
```

[104]:
```python
# Understand the data shape
raw_data.shape
```

[104]:
```
(255347, 18)
```

[110]:
```python
# There are 255347 observations and 18 columns in the dataset
```

[112]:
```python
### Check the data types of the columns in the dataset.
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 255347 entries, 0 to 255346
Data columns (total 18 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   LoanID          255347 non-null  object
 1   Age             255347 non-null  int64
 2   Income          255347 non-null  int64
 3   LoanAmount      255347 non-null  int64
 4   CreditScore     255347 non-null  int64
 5   MonthsEmployed  255347 non-null  int64
 6   NumCreditLines  255347 non-null  int64
 7   InterestRate    255347 non-null  float64
 8   LoanTerm        255347 non-null  int64
 9   DTIRatio        255347 non-null  float64
 10  Education       255347 non-null  object
 11  EmploymentType  255347 non-null  object
 12  MaritalStatus   255347 non-null  object
 13  HasMortgage     255347 non-null  object
 14  HasDependents   255347 non-null  object
 15  LoanPurpose     255347 non-null  object
 16  HasCoSigner     255347 non-null  object
 17  Default         255347 non-null  int64
```

```
dtypes: float64(2), int64(8), object(8)
memory usage: 35.1+ MB
```

[115]:
```
###Summary of the data
raw_data.describe().T
```

[115]:
```
                   count           mean           std      min       25%  \
Age             255347.0      43.498306     14.990258     18.0     31.00
Income          255347.0   82499.304597  38963.013729  15000.0  48825.50
LoanAmount      255347.0  127578.865512  70840.706142   5000.0  66156.00
CreditScore     255347.0     574.264346    158.903867    300.0    437.00
MonthsEmployed  255347.0      59.541976     34.643376      0.0     30.00
NumCreditLines  255347.0       2.501036      1.117018      1.0      2.00
InterestRate    255347.0      13.492773      6.636443      2.0      7.77
LoanTerm        255347.0      36.025894     16.969330     12.0     24.00
DTIRatio        255347.0       0.500212      0.230917      0.1      0.30
Default         255347.0       0.116128      0.320379      0.0      0.00

                      50%        75%       max
Age                 43.00      56.00      69.0
Income           82466.00  116219.00  149999.0
LoanAmount      127556.00  188985.00  249999.0
CreditScore        574.00     712.00     849.0
MonthsEmployed      60.00      90.00     119.0
NumCreditLines       2.00       3.00       4.0
InterestRate        13.46      19.25      25.0
LoanTerm            36.00      48.00      60.0
DTIRatio             0.50       0.70       0.9
Default              0.00       0.00       1.0
```

[116]:
```
# Wide range, most borrowers are adults across generations.
# Moderate variation - outliers possible (check skew).
# High standard deviation = large loan variability.
# Covers full credit score spectrum; higher = less risk.
# Most borrowers have steady work history; 0 = unemployed.
# Limited credit history (most have 2-3 accounts).
# High variance - riskier borrowers get higher rates.
# Most loans are short to medium term (3-5 years).
# Normal range; 0.43+ may be flagged as high risk.
# Class imbalance: only ~12% are defaulters.
```

[119]:
```
###Display number of missing values per each column
raw_data.isna().sum()
```

[119]:
```
LoanID            0
Age               0
Income            0
LoanAmount        0
```

```
CreditScore       0
MonthsEmployed    0
NumCreditLines    0
InterestRate      0
LoanTerm          0
DTIRatio          0
Education         0
EmploymentType    0
MaritalStatus     0
HasMortgage       0
HasDependents     0
LoanPurpose       0
HasCoSigner       0
Default           0
dtype: int64
```

[121]:
```
# Your dataset has no missing values in any column.
```

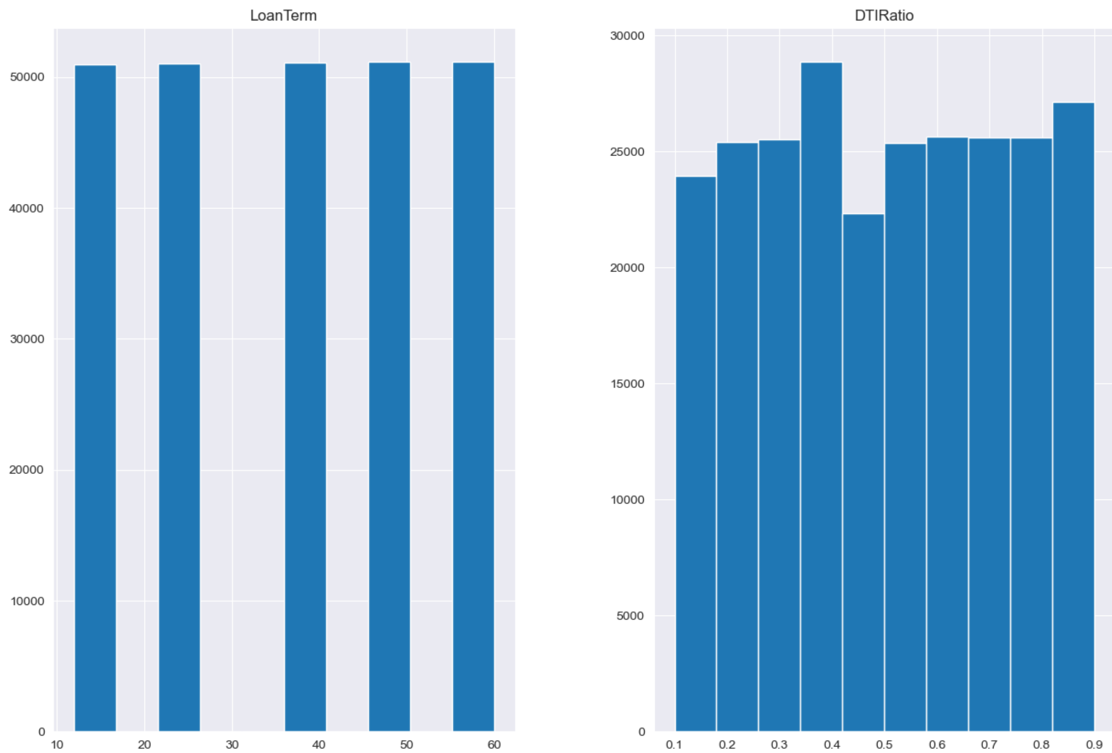
[123]:
```
# check for duplicate rows based on all columns
duplicate_rows = raw_data[raw_data.duplicated()]
print(duplicate_rows)
```

```
Empty DataFrame
Columns: [LoanID, Age, Income, LoanAmount, CreditScore, MonthsEmployed,
NumCreditLines, InterestRate, LoanTerm, DTIRatio, Education, EmploymentType,
MaritalStatus, HasMortgage, HasDependents, LoanPurpose, HasCoSigner, Default]
Index: []
```

[124]:
```
# There are no  duplicates
```

[125]:
```
data = raw_data.copy()
```

[126]:
```
## Univariate analysis
sns.set_style("darkgrid")
data.iloc[:, 8:17].hist(figsize=(15, 10))
plt.show()
```



[130]:
```
data.iloc[:, 8:17].boxplot(figsize=(20, 5))
plt.show()
```

[131]:
```
# Checking the loan Status distibution amoung defaults and non defaults
data["Default"].value_counts(1)
```

[131]:
```
Default
0    0.883872
1    0.116128
Name: proportion, dtype: float64
```

[132]:
```python
# Only 11.6% of borrowers defaulted.
```

[133]:
```python
# Cross-table of gender and Loan Status
round(data.groupby(["Age"])["Default"].value_counts(1), 2).unstack()
```

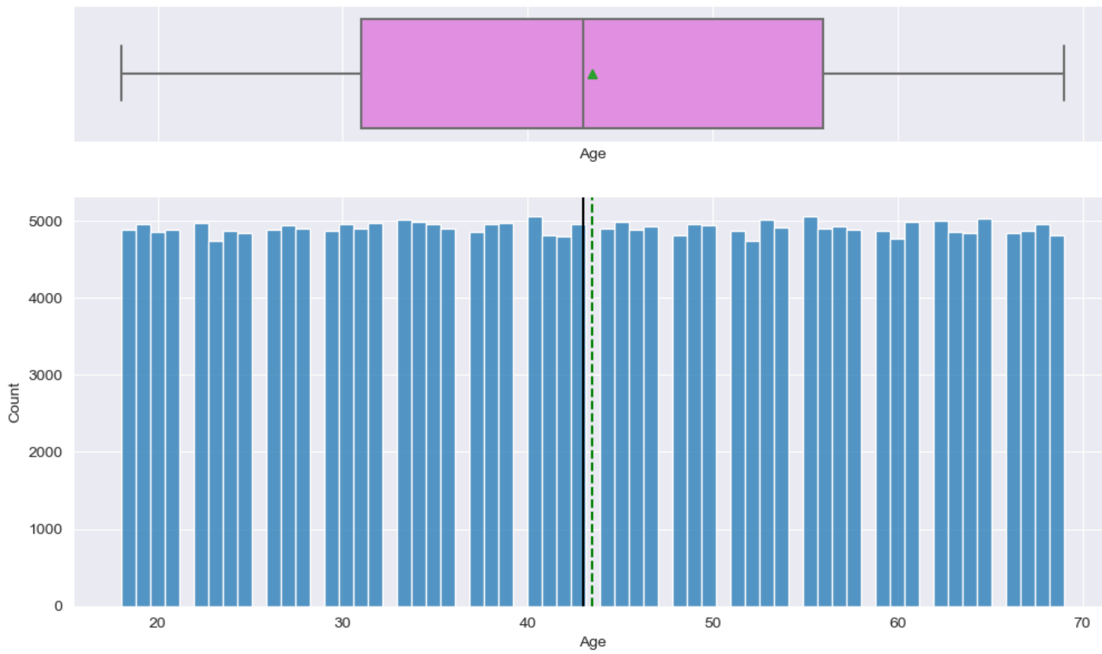
[133]:

Default	0	1
Age		
18	0.78	0.22
19	0.78	0.22
20	0.78	0.22
21	0.80	0.20
22	0.78	0.22
23	0.81	0.19
24	0.81	0.19
25	0.81	0.19
26	0.82	0.18
27	0.82	0.18
28	0.82	0.18
29	0.84	0.16
30	0.84	0.16
31	0.84	0.16
32	0.84	0.16
33	0.85	0.15
34	0.86	0.14
35	0.86	0.14
36	0.86	0.14
37	0.87	0.13
38	0.87	0.13
39	0.89	0.11
40	0.89	0.11
41	0.89	0.11
42	0.89	0.11
43	0.90	0.10
44	0.90	0.10
45	0.90	0.10
46	0.90	0.10
47	0.91	0.09
48	0.91	0.09
49	0.91	0.09
50	0.91	0.09
51	0.92	0.08
52	0.92	0.08
53	0.93	0.07
54	0.92	0.08
55	0.93	0.07
56	0.93	0.07
57	0.94	0.06

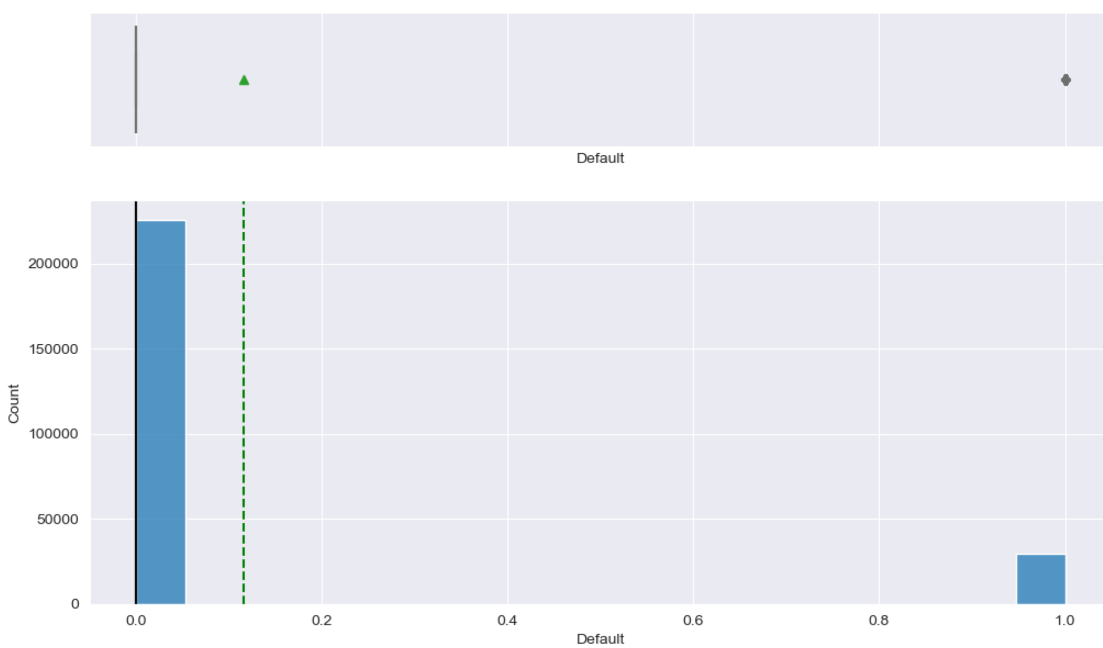
```
58       0.94  0.06
59       0.94  0.06
60       0.94  0.06
61       0.95  0.05
62       0.95  0.05
63       0.95  0.05
64       0.94  0.06
65       0.94  0.06
66       0.96  0.04
67       0.95  0.05
68       0.95  0.05
69       0.96  0.04
```

[134]:
```python
def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """

    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the␣
↪column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

[136]:
```
### Observation on Age
histogram_boxplot(data, "Age")
```



[137]:
```
histogram_boxplot(data, "Default")
```



[138]:
```
data.head(5)
```

[138]:
```
       LoanID  Age  Income  LoanAmount  CreditScore  MonthsEmployed  \
0  I38PQUQS96   56   85994       50587          520              80
1  HPSK72WA7R   69   50432      124440          458              15
2  C1OZ6DPJ8Y   46   84208      129188          451              26
3  V2KKSFM3UN   32   31713       44799          743               0
4  EY08JDHTZP   60   20437        9139          633               8

   NumCreditLines  InterestRate  LoanTerm  DTIRatio    Education  \
0               4         15.23        36      0.44   Bachelor's
1               1          4.81        60      0.68     Master's
2               3         21.17        24      0.31     Master's
3               3          7.07        24      0.23  High School
4               4          6.51        48      0.73   Bachelor's

  EmploymentType MaritalStatus HasMortgage HasDependents LoanPurpose  \
0      Full-time      Divorced         Yes           Yes       Other
1      Full-time       Married          No            No       Other
2     Unemployed      Divorced         Yes           Yes        Auto
3      Full-time       Married          No            No    Business
4     Unemployed      Divorced          No           Yes        Auto

  HasCoSigner  Default
0         Yes        0
1         Yes        0
2          No        1
3          No        0
4          No        0
```

X = data.drop(columns=["LoanID", "Default"]) Y = data["Default"]

[139]:
```
# 1. Split into X and Y
X = data.drop(columns=["LoanID", "Default"])
Y = data["Default"]

# 2. Encode categorical variables
X = pd.get_dummies(X, drop_first=True)

# 3. Scale numerical columns
scaler = StandardScaler()
num_cols = [
    "Age",
    "Income",
    "LoanAmount",
    "CreditScore",
    "MonthsEmployed",
    "NumCreditLines",
```

```
    "InterestRate",
    "LoanTerm",
    "DTIRatio",
]
X[num_cols] = scaler.fit_transform(X[num_cols])

# 4. Save the scaler
pickle.dump(
    scaler,
    open(
        "C:/Users/Munashe Muchinako/OneDrive/Desktop/data science/kraddle proj/
↪Models/StandardScaler.pkl",
        "wb",
    ),
)

# 5. Add constant for statsmodels
X = sm.add_constant(X)

# 6. Split into Train, Validation, and Test sets
X_temp, X_test, y_temp, y_test = train_test_split(
    X, Y, test_size=0.2, stratify=Y, random_state=42
)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.2, stratify=y_temp, random_state=42
)

# 7. Output shape check
print("Train:", X_train.shape)
print("Validation:", X_val.shape)
print("Test:", X_test.shape)
```

```
Train: (163421, 25)
Validation: (40856, 25)
Test: (51070, 25)
```

[140]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
```

```
)

# Define models
models = {
    "Logistic Regression": LogisticRegression(
        max_iter=1000, class_weight="balanced", random_state=42
    ),
    "Random Forest": RandomForestClassifier(
        n_estimators=100, class_weight="balanced", random_state=42
    ),
    "XGBoost": XGBClassifier(
        use_label_encoder=False, eval_metric="logloss", random_state=42
    ),
    "Decision Tree": DecisionTreeClassifier(class_weight="balanced",␣
↪random_state=42),
}


# Function to evaluate a model
def evaluate_model(model, X_val, y_val):
    y_pred = model.predict(X_val)
    y_prob = (
        model.predict_proba(X_val)[:, 1] if hasattr(model, "predict_proba")␣
↪else None
    )

    return {
        "Accuracy": accuracy_score(y_val, y_pred),
        "Precision": precision_score(y_val, y_pred),
        "Recall": recall_score(y_val, y_pred),
        "F1 Score": f1_score(y_val, y_pred),
        "ROC AUC": roc_auc_score(y_val, y_prob) if y_prob is not None else "N/
↪A",
    }


# Train and evaluate all models
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    metrics = evaluate_model(model, X_val, y_val)
    results[name] = metrics
    print(f"\n {name} Results:")
    for metric, score in metrics.items():
        print(
            f"{metric}: {score:.4f}"
            if isinstance(score, float)
```

```
            else f"{metric}: {score}"
        )
```

```
 Logistic Regression Results:
Accuracy: 0.6727
Precision: 0.2140
Recall: 0.6804
F1 Score: 0.3256
ROC AUC: 0.7420

 Random Forest Results:
Accuracy: 0.8849
Precision: 0.6408
Recall: 0.0192
F1 Score: 0.0372
ROC AUC: 0.7250

 XGBoost Results:
Accuracy: 0.8845
Precision: 0.5159
Recall: 0.0822
F1 Score: 0.1418
ROC AUC: 0.7333

 Decision Tree Results:
Accuracy: 0.8152
Precision: 0.1958
Recall: 0.1903
F1 Score: 0.1931
ROC AUC: 0.5438
```

[142]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    "C": [0.01, 0.1, 1, 10],
    "penalty": ["l1", "l2"],
    "solver": ["liblinear"],  # works with both l1 and l2
}

# Set up grid search
grid = GridSearchCV(
    LogisticRegression(class_weight="balanced", max_iter=1000),
    param_grid,
    scoring="f1",
```

```
    cv=5,
    n_jobs=-1,
    verbose=1,
)

# Fit the model
grid.fit(X_train, y_train)

# Best model
best_logreg = grid.best_estimator_

# Evaluate on validation set
from sklearn.metrics import classification_report, roc_auc_score

y_pred = best_logreg.predict(X_val)
y_prob = best_logreg.predict_proba(X_val)[:, 1]

print("\n  Best Logistic Regression Results:")
print(classification_report(y_val, y_pred))
print(f"ROC AUC: {roc_auc_score(y_val, y_prob):.4f}")
print(f"Best Params: {grid.best_params_}")
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits

  Best Logistic Regression Results:
              precision    recall  f1-score   support

           0       0.94      0.67      0.78     36112
           1       0.21      0.68      0.33      4744

    accuracy                           0.67     40856
   macro avg       0.58      0.68      0.55     40856
weighted avg       0.86      0.67      0.73     40856

ROC AUC: 0.7419
Best Params: {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
```

[147]:

```
###Show Top Features (L1 Weights)
# Get feature names
feature_names = X_train.columns

# Get coefficients from the model
coefficients = best_logreg.coef_[0]

# Combine into a DataFrame
coef_df = pd.DataFrame({"Feature": feature_names, "Coefficient": coefficients})
```

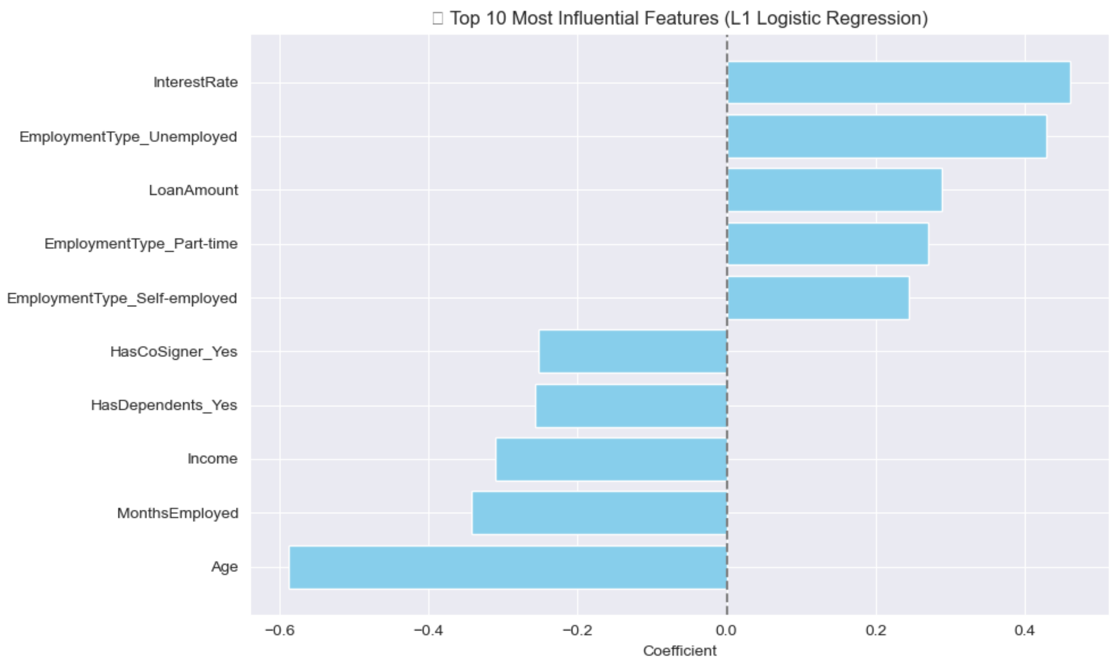
```python
# Remove zero-weighted features (dropped by L1)
non_zero = coef_df[coef_df["Coefficient"] != 0]

# Sort by absolute importance
non_zero["AbsCoefficient"] = np.abs(non_zero["Coefficient"])
top_features = non_zero.sort_values(by="AbsCoefficient", ascending=False)

# Display top 10 most influential features
print("\n Top 10 Most Influential Features (L1 Regularized):")
print(top_features[["Feature", "Coefficient"]].head(10))
```

```
 Top 10 Most Influential Features (L1 Regularized):
                        Feature  Coefficient
1                           Age    -0.587940
7                  InterestRate     0.460850
15     EmploymentType_Unemployed     0.430141
5                MonthsEmployed    -0.341973
2                        Income    -0.310661
3                    LoanAmount     0.288986
13      EmploymentType_Part-time     0.270401
19             HasDependents_Yes    -0.256662
24               HasCoSigner_Yes    -0.252747
14  EmploymentType_Self-employed     0.244065
```

Rank	Feature	Coefficient	Insight
1	`Age`	-0.59	Older borrowers are **less likely to default** (protective factor)
2	`InterestRate`	+0.46	Higher rates → **more likely to default** (often a proxy for higher risk loans)
3	`EmploymentType_Unemployed`	+0.43	Strong risk factor — unemployment correlates with **default**
4	`MonthsEmployed`	-0.34	More months employed → **more stability** → less default
5	`Income`	-0.31	Higher income is **protective**
6	`LoanAmount`	+0.29	Larger loans → higher risk of **default**
7	`EmploymentType_Part-time`	+0.27	Part-time workers show increased risk
8	`HasDependents_Yes`	-0.26	Interesting: may indicate **more responsible borrowers**, or credit bias
9	`HasCoSigner_Yes`	-0.25	A co-signer reduces risk (added creditworthiness)
	`EmploymentType_Self-employed`	+0.24	Self-employed = higher uncertainty → more risk

[149]:
```python
# Sort top 10 features
top10 = top_features[["Feature", "Coefficient"]].head(10)
top10 = top10.sort_values(by="Coefficient")

# Plot
```

```
plt.figure(figsize=(10, 6))
plt.barh(top10["Feature"], top10["Coefficient"], color="skyblue")
plt.axvline(0, color="gray", linestyle="--")
plt.title("  Top 10 Most Influential Features (L1 Logistic Regression)")
plt.xlabel("Coefficient")
plt.tight_layout()
plt.show()
```



[ ]:

```
[151]: # Use the top 10 feature names (from L1)
top_feature_names = top_features["Feature"].head(10).tolist()

# Add 'const' to match model input
top_feature_names_with_const = ["const"] + top_feature_names

# Prepare dataset
X_top = X[top_feature_names]
X_top = sm.add_constant(X_top)

# Train/test split
X_train_top, X_val_top, y_train_top, y_val_top = train_test_split(
    X_top, Y, test_size=0.2, stratify=Y, random_state=42
)
```

```
# Train the model
top_model = LogisticRegression(
    C=0.1, penalty="l1", solver="liblinear", class_weight="balanced",␣
 ↪max_iter=1000
)
top_model.fit(X_train_top, y_train_top)

# Save the model
joblib.dump(
    top_model,
    "C:/Users/Munashe Muchinako/OneDrive/Desktop/data science/kraddle proj/
 ↪Models/logistic_top_model.pkl",
)

# Save the corrected list of feature names
with open(
    "C:/Users/Munashe Muchinako/OneDrive/Desktop/data science/kraddle proj/
 ↪Models/top_model_features.pkl",
    "wb",
) as f:
    pickle.dump(top_feature_names_with_const, f)
```

[153]:

```
# Load model and features
model = joblib.load(
    "C:/Users/Munashe Muchinako/OneDrive/Desktop/data science/kraddle proj/
 ↪Models/logistic_top_model.pkl"
)
with open(
    "C:/Users/Munashe Muchinako/OneDrive/Desktop/data science/kraddle proj/
 ↪Models/top_model_features.pkl",
    "rb",
) as f:
    feature_names = pickle.load(f)

# Get non-zero coefficients
coefficients = model.coef_[0]
importance_df = pd.DataFrame(
    {
        "feature": feature_names,
        "coefficient": coefficients,
        "abs_importance": np.abs(coefficients),
    }
).sort_values(by="abs_importance", ascending=False)

# Save to JSON
importance_df.to_json(
```

```
    "C:/Users/Munashe Muchinako/OneDrive/Desktop/data science/kraddle proj/
 ↪Models/feature_importance.json",
    orient="records",
)
```

[170]:
```
# Predict
y_pred = model.predict(X_val_top)
y_proba = model.predict_proba(X_val_top)[:, 1]

# Calculate metrics
kpis = {
    "default_rate": round(y_val_top.sum() / len(y_val_top), 4),
    "accuracy": round(accuracy_score(y_val_top, y_pred), 4),
    "precision": round(precision_score(y_val_top, y_pred), 4),
    "recall": round(recall_score(y_val_top, y_pred), 4),
    "f1_score": round(f1_score(y_val_top, y_pred), 4),
    "roc_auc": round(roc_auc_score(y_val_top, y_proba), 4),
}

# Save to JSON
with open(
    "C:/Users/Munashe Muchinako/OneDrive/Desktop/data science/kraddle proj/
 ↪Models/kpi_metrics.json",
    "w",
) as f:
    json.dump(kpis, f)
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: