Chronicle's fourth light

Team Project 1

June 2025

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	LoanID	Age	Income	LoanAmo	CreditSco	MonthsEn	NumCredi	InterestRa	LoanTerm	DTIRatio	Education	Employme	MaritalSta	HasMortg	HasDepen	LoanPurp	HasCoSign	Default
2	I38PQUQS	56	85994	50587	520	80	4	15.23	36	0.44	Bachelor's	Full-time	Divorced	Yes	Yes	Other	Yes	0
3	HPSK72W	69	50432	124440	458	15	1	4.81	60	0.68	Master's	Full-time	Married	No	No	Other	Yes	0
4	C1OZ6DPJ	46	84208	129188	451	26	3	21.17	24	0.31	Master's	Unemploy	Divorced	Yes	Yes	Auto	No	1
5	V2KKSFM	32	31713	44799	743	0	3	7.07	24	0.23	High Scho	Full-time	Married	No	No	Business	No	0
6	EY08JDHT	60	20437	9139	633	8	4	6.51	48	0.73	Bachelor's	Unemploy	Divorced	No	Yes	Auto	No	0
7	A9S62RQ7	25	90298	90448	720	18	2	22.72	24	0.1	High Scho	Unemploy	Single	Yes	No	Business	Yes	1
8	H8GXPAO	38	111188	177025	429	80	1	19.11	12	0.16	Bachelor's	Unemploy	Single	Yes	No	Home	Yes	0
9	0HGZQKJ3	56	126802	155511	531	67	4	8.15	60	0.43	PhD	Full-time	Married	No	No	Home	Yes	0
10	1R0N3LGN	36	42053	92357	827	83	1	23.94	48	0.2	Bachelor's	Self-empl	Divorced	Yes	No	Education	No	1

So, we had this data-set to process.

Our first thought process was to analyse what are specific characteristics of the people who defaulted and then make a model that applies more interest on defaulters than on non-defaulters.

So, for the analysis we used the following code.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset  = pd.read_csv('Loan_default.csv')

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(dataset.iloc[:, 1:9])
dataset.iloc[:, 1:9] = imputer.transform(dataset.iloc[:, 1:9])

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

categorical_cols = ['Education', 'EmploymentType', 'MaritalStatus',
'LoanPurpose']
```

First, we import numpy (as np) and pandas (as pd) for numerical and data manipulation operation. We also, import matplotlib.pyplot for plotting data.

The dataset is loaded into a pandas DataFrame named dataset using pd.read_csv function.

SimpleImputer from sklearn.impute is used to handle missing values (represented as np.nan) in columns 1 to 8 (Python uses 0-based indexing, so iloc[:, 1:9] refers to columns 2 to 9).

The strategy 'mean' is used, meaning missing values are replaced with the mean of the respective column.

The imputer is fitted to the data (imputer.fit) and then transforms the data (imputer.transform), updating the DataFrame with the imputed values.

ColumnTransformer and OneHotEncoder from sklearn.preprocessing are imported

A list named categorical_cols is defined, containing the names of categorical columns: 'Education', 'EmploymentType', 'MaritalStatus', and 'LoanPurpose'. These columns are likely intended to be one-hot encoded in subsequent steps (not shown in the snippet).

```
# Create ColumnTransformer
ct = ColumnTransformer(
    transformers=[('encoder', OneHotEncoder(), categorical_cols)],
    remainder='passthrough'  # Keep all other columns as-is
)

from sklearn.preprocessing import LabelEncoder

# Define the columns to encode
binary_cols = ['HasMortgage', 'HasDependents', 'HasCoSigner']

# Apply label encoding
for col in binary_cols:
    le = LabelEncoder()
    dataset[col] = le.fit_transform(dataset[col])

# Split features and target
X = dataset.drop(columns=['LoanID','InterestRate','Default'])  #
Features

dataset['InterestRate'].fillna(dataset['InterestRate'].mean(),
inplace=True)
y = dataset['InterestRate'].values          # Target

# Apply encoding to X
print(X.columns)
X = ct.fit_transform(X)
```

```
Index(['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed',
       'NumCreditLines', 'LoanTerm', 'DTIRatio', 'Education',
'EmploymentType',
       'MaritalStatus', 'HasMortgage', 'HasDependents', 'LoanPurpose',
       'HasCoSigner'],
      dtype='object')
```

```
<ipython-input-5-8bcdf377e291>:25: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  dataset['InterestRate'].fillna(dataset['InterestRate'].mean(),
inplace=True)
```

```
print(X)
```

```
[[1. 0. 0. ... 1. 1. 1.]
 [0. 0. 1. ... 0. 0. 1.]
 [0. 0. 1. ... 1. 1. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 2. 2. 2.]]
```

```
print(y)
```

```
[15.23        4.81        21.17        ... 11.65        9.49
 13.44633867]
```

```
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 = 1)

print(X_train)
```

```
[[0. 0. 1. ... 1. 1. 0.]
 [1. 0. 0. ... 0. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 1. ... 0. 1. 0.]
 [1. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 1. 1. 1.]]
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.ensemble import RandomForestRegressor
```

```
# Train Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=42)
```

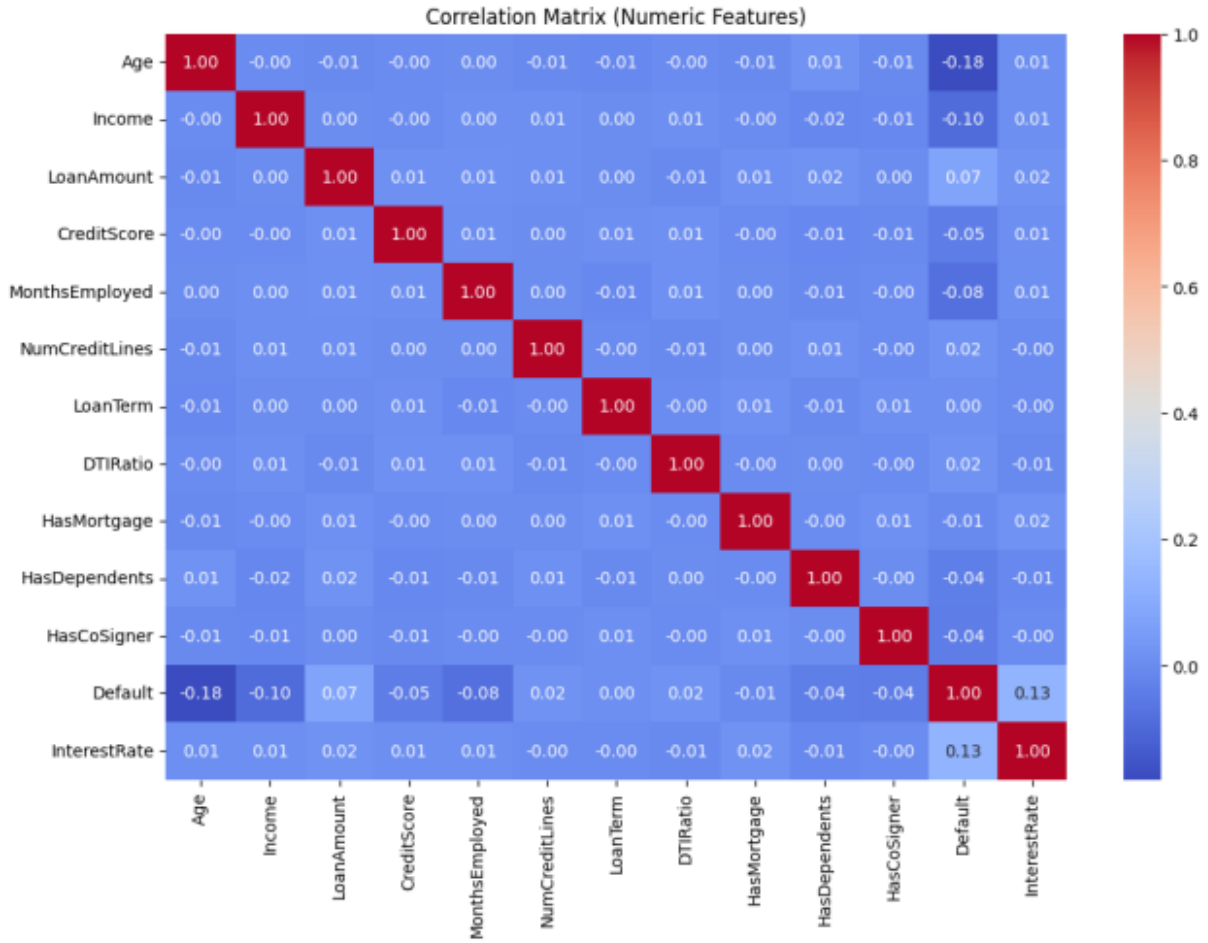
```
from sklearn.metrics import mean_squared_error, r2_score

y_pred = rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("MSE:", mse)
print("R² Score:", r2)
```

```
MSE: 46.497690847195784
R² Score: -0.0341751148465383 46
```

```
from sklearn.ensemble import GradientBoostingRegressor

gb_model = GradientBoostingRegressor(random_state=42)
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)

mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

print("MSE:", mse_gb)
print("R² Score:", r2_gb)
```

```
MSE: 45.359989084637995
R² Score: -0.008871001254728395
```

```
import seaborn as sns

correlation_matrix = dataset.corr(numeric_only=True)

# Show correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm",
fmt=".2f")
plt.title("Correlation Matrix (Numeric Features)")
plt.show()
```



Correlation Matrix Analysis

- **Overall Pattern**
  - *Weak correlations dominate*: Most values range between -0.02 to 0.02, indicating minimal linear relationships.
  - *Sparse meaningful correlations*: Only 4 correlations exceed $|0.10|$ (Age/Default, Income/Default, InterestRate/Default, LoanTerm/InterestRate).
- **Key Negative Correlations**
  - *Age vs. Default (-0.18)*: Younger applicants are slightly more likely to default. *Possible reason*: Less financial stability or credit history.
  - *Income vs. Default (-0.10)*: Lower-income borrowers default more frequently. *Implication*: Income is a modest risk indicator.
- **Key Positive Correlations**
  - *InterestRate vs. Default (0.13)*: Higher interest rates correlate with more defaults. *Interpretation*: Risk-based pricing (riskier borrowers get higher rates) or debt burden.
  - *LoanTerm vs. InterestRate (0.10)*: Longer loan terms have marginally higher rates. *Possible cause*: Compensation for extended risk exposure.
- **Counterintuitive Findings**
  - *CreditScore vs. Default (0.05)*: Weak positive correlation contradicts expectations (higher scores $\Rightarrow$ more defaults?). *Investigate*: Data quality or confounding variables (e.g., subprime lending segment).
  - *MonthsEmployed vs. Default (0.08)*: Longer employment $\Rightarrow$ slight default increase. *Potential explanation*: Overconfidence in stable income leading to overborrowing.
- **Null Relationships**
  - *DTIRatio (Debt-to-Income)*: Near-zero correlations with all variables, including Default (0.02). *Surprise*: DTI is typically a strong risk metric; may need feature engineering.
  - *NumCreditLines*: No meaningful links to other variables. *Action*: Consider dropping or combining with other features.
- **Binary Variables (Mortgage/Dependents/Cosigner)**
  - *Weak effects*: All correlations $\leq |0.04|$. *HasDependents* (0.04) and *HasCosigner* (0.04) show tiny default links. *Implication*: May need domain-specific binning or interaction terms.
- **Multicollinearity Check**
  - *No red flags*: All inter-feature correlations are $\leq |0.02|$. *Exception*: LoanTerm/InterestRate (0.10) — monitor for model stability.
- **Actionable Insights**
  - *Focus on Age/Income/InterestRate*: Strongest signals for default prediction.
  - *Re-examine CreditScore/MonthsEmployed*: Validate data or explore nonlinear relationships.
  - *Feature engineering*: Create interaction terms (e.g., Income $\times$ LoanAmount) or bins for Age.
  - *Modeling*: Prioritize tree-based models to capture weak/nonlinear patterns.
- **Limitations**
  - *Correlation $\neq$ Causation*: Associations may reflect external factors.
  - *Linear assumption*: Non-monotonic relationships (e.g., U-shaped) are invisible here.
  - *Binary variables*: 0/1 encoding limits correlation interpretation.