Lab 8 - Classification - ML - Credit dataset

https://www.kaggle.com/datasets/uciml/german-credit https://medium.com/swlh/german-credit-risk-classification-modeling-and-metrics-19182b87f060 https://machinelearningmastery.com/imbalanced-classification-of-good-and-bad-credit/

```
seed = 42
```

Loading the data

```
# Option 1
# Download the CSV from https://www.kaggle.com/datasets/uciml/german-credit and load it using pandas
#import pandas as pd
#df = pd.read_csv('german_credit.csv')

# Option 2
# Some datasets are already included with sklearn for teaching/exemplification purposes: https://scikit-learn.org/stable/ap
# Others can be downloaded using fetch_openml from openml (https://www.openml.org/)
# Let's serach for "german credit". The first entry should https://www.openml.org/search?type=data&status=active&id=31
from sklearn.datasets import fetch_openml
#credit = fetch_openml(name='credit-g', as_frame=True)
# as_frame: If True, the data is a pandas DataFrame including columns with appropriate dtypes
# This dataset has several versions (revisions). We can choose a specific version, by specifying the "version" parameter.
credit = fetch_openml(name='credit-g', version = 1, as_frame=True)
df = credit.frame
```

Let's check the dataset

```
df.head()
```

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commi
0	<0	6	critical/other existing credit	radio/tv	1169	no known savings	>=7	
1	0<=X<200	48	existing paid	radio/tv	5951	<100	1<=X<4	
2	no checking	12	critical/other existing credit	education	2096	<100	4<=X<7	
3	<0	42	existing paid	furniture/equipment	7882	<100	4<=X<7	
4	<0	24	delayed previously	new car	4870	<100	1<=X<4	

5 rows × 21 columns

```
df.describe()
```

	duration	credit_amount	installment_commitment	residence_since	age	existing_credits	num_dependents
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	20.903000	3271.258000	2.973000	2.845000	35.546000	1.407000	1.155000
std	12.058814	2822.736876	1.118715	1.103718	11.375469	0.577654	0.362086
min	4.000000	250.000000	1.000000	1.000000	19.000000	1.000000	1.000000
25%	12.000000	1365.500000	2.000000	2.000000	27.000000	1.000000	1.000000
50%	18.000000	2319.500000	3.000000	3.000000	33.000000	1.000000	1.000000
75%	24.000000	3972.250000	4.000000	4.000000	42.000000	2.000000	1.000000
max	72.000000	18424.000000	4.000000	4.000000	75.000000	4.000000	2.000000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   checking_status       1000 non-null   category
 1   duration              1000 non-null   int64
```

```
 2   credit_history          1000 non-null   category
 3   purpose                 1000 non-null   category
 4   credit_amount           1000 non-null   int64
 5   savings_status          1000 non-null   category
 6   employment              1000 non-null   category
 7   installment_commitment  1000 non-null   int64
 8   personal_status         1000 non-null   category
 9   other_parties           1000 non-null   category
 10  residence_since         1000 non-null   int64
 11  property_magnitude      1000 non-null   category
 12  age                     1000 non-null   int64
 13  other_payment_plans     1000 non-null   category
 14  housing                 1000 non-null   category
 15  existing_credits        1000 non-null   int64
 16  job                     1000 non-null   category
 17  num_dependents          1000 non-null   int64
 18  own_telephone           1000 non-null   category
 19  foreign_worker          1000 non-null   category
 20  class                   1000 non-null   category
dtypes: category(14), int64(7)
memory usage: 71.0 KB
```

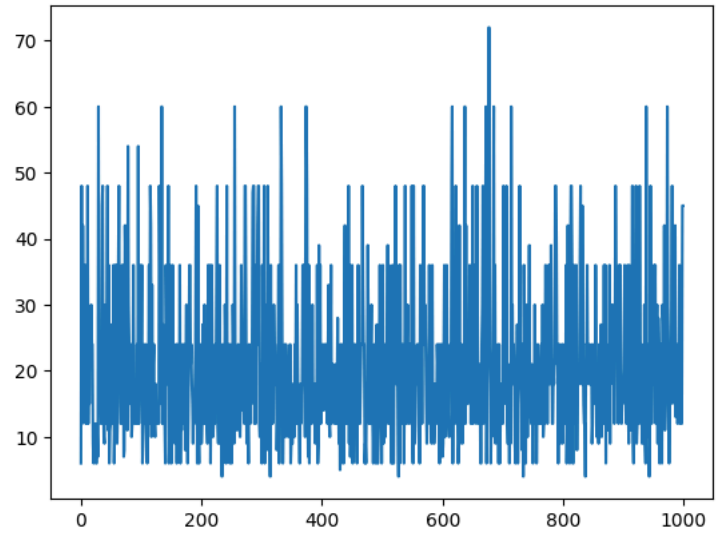
```
df.head(2)
```

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitment	pe
0	<0	6	critical/other existing credit	radio/tv	1169	no known savings	>=7	4	
1	0<=X<200	48	existing paid	radio/tv	5951	<100	1<=X<4	2	

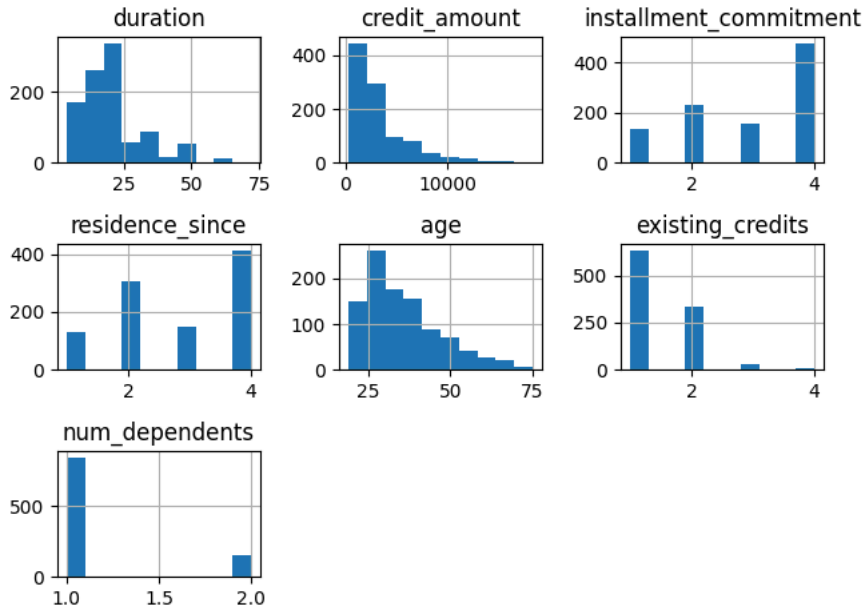
2 rows × 21 columns

```
df['duration'].plot()
```

```
<Axes: >
```




```
import matplotlib.pyplot as plt
# select only the columns with numeric values
numeric_columns = df.select_dtypes(include=['number'])
ax = numeric_columns.hist()
plt.tight_layout()
plt.show()
```



Let's check if the classes ('good' and 'bad') are balanced.

```
df['class'].value_counts()
```

class	count
good	700
bad	300

dtype: int64

```
df['class'].value_counts(normalize=True)
```

class	proportion
good	0.7
bad	0.3

dtype: float64

```
df_good = df[df['class'] == 'good']
df_good_sampled = df_good.sample(300, random_state=seed)

df_bad_sampled = df[df['class'] == 'bad']

df_balanced = pd.concat([df_good_sampled, df_bad_sampled])
df_balanced.shape
```

```
(600, 21)
```

Classification

```
X = df_balanced.drop('class', axis=1)
y = df_balanced['class'] # good and bad
```

```
# Convert categorical columns
# https://www.geeksforgeeks.org/machine-learning/ml-one-hot-encoding/
import pandas as pd
X = pd.get_dummies(X)
X
```

	duration	credit_amount	installment_commitment	residence_since	age	existing_credits	num_dependents	checking_statu
218	24	3021	2	2	24	1	1	
712	21	2476	4	4	46	1	1	
550	12	996	4	4	23	2	1	
215	6	932	1	3	39	2	1	
441	12	1620	2	3	30	1	1	
...	...	...	...	...	...	...	...	
979	15	1264	2	2	25	1	1	
980	30	8386	2	2	49	1	1	
981	48	4844	3	2	33	1	1	
983	36	8229	2	2	26	1	2	
998	45	1845	4	4	23	1	1	

600 rows × 61 columns

```
# Normalize data using StandardScaler
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# Reason: many elements used in the objective function of a learning algorithm (such as the RBF kernel of
# Support Vector Machines or the L1 and L2 regularizers of linear models) assume that all features are
# centered around 0 and have variance in the same order. If a feature has a variance that is orders of
# magnitude larger than others, it might dominate the objective function and make the estimator unable to learn
# from other features correctly as expected.
# Note: Might not be needed for all classification models

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

```
array([[ 0.13949417, -0.15372759, -0.9031329 , ..., -0.79959006,
        -0.18569534,  0.18569534],
       [-0.09827998, -0.33278686,  0.8941465 , ...,  1.25064086,
        -0.18569534,  0.18569534],
       [-0.81160243, -0.81903956,  0.8941465 , ..., -0.79959006,
        -0.18569534,  0.18569534],
       ...,
       [ 2.04168736,  0.44521747, -0.0044932 , ...,  1.25064086,
        -0.18569534,  0.18569534],
       [ 1.09059076,  1.55735625, -0.9031329 , ..., -0.79959006,
        -0.18569534,  0.18569534],
       [ 1.80391321, -0.54010136,  0.8941465 , ...,  1.25064086,
        -0.18569534,  0.18569534]])
```

```
# The .length attribute does not exist for numpy arrays, use .shape to get the dimensions.
print(X_scaled.shape)
```

```
(600, 61)
```

```
# Split teh data in trainning and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=seed)
```

```
# Train a LogisticRegression classifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_repor

# Model Initialization and Training
model = LogisticRegression(random_state=seed)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Display the metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, pos_label='bad'))
print("Recall:", recall_score(y_test, y_pred, pos_label='bad'))
print("F1 Score:", f1_score(y_test, y_pred, pos_label='bad'))
```

```
class_names = ['good', 'bad']
print(classification_report(y_test, y_pred, target_names=class_names))
```

```
Accuracy: 0.675
Precision: 0.6507936507936508
Recall: 0.7068965517241379
F1 Score: 0.6776859504132231
              precision    recall  f1-score   support

        good       0.65      0.71      0.68        58
         bad       0.70      0.65      0.67        62

    accuracy                           0.68       120
   macro avg       0.68      0.68      0.67       120
weighted avg       0.68      0.68      0.67       120
```

```
# Train a RandomForestClassifier classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_repor

# Model Initialization and Training
model = RandomForestClassifier(random_state=seed)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Display the metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, pos_label='bad'))
print("Recall:", recall_score(y_test, y_pred, pos_label='bad'))
print("F1 Score:", f1_score(y_test, y_pred, pos_label='bad'))

class_names = ['good', 'bad']
print(classification_report(y_test, y_pred, target_names=class_names))
```

```
Accuracy: 0.6583333333333333
Precision: 0.6231884057971014
Recall: 0.7413793103448276
F1 Score: 0.6771653543307087
              precision    recall  f1-score   support

        good       0.62      0.74      0.68        58
         bad       0.71      0.58      0.64        62

    accuracy                           0.66       120
   macro avg       0.66      0.66      0.66       120
weighted avg       0.67      0.66      0.66       120
```

```
# Train a LinearSVC classifier
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_repor

# Model Initialization and Training
model = LinearSVC(random_state=seed)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Display the metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, pos_label='bad'))
print("Recall:", recall_score(y_test, y_pred, pos_label='bad'))
print("F1 Score:", f1_score(y_test, y_pred, pos_label='bad'))

class_names = ['good', 'bad']
print(classification_report(y_test, y_pred, target_names=class_names))
```

```
Accuracy: 0.6833333333333333
Precision: 0.65625
Recall: 0.7241379310344828
F1 Score: 0.6885245901639344
              precision    recall  f1-score   support

        good       0.66      0.72      0.69        58
         bad       0.71      0.65      0.68        62

    accuracy                           0.68       120
   macro avg       0.69      0.68      0.68       120
weighted avg       0.69      0.68      0.68       120
```

```
# Train a MLPClassifier classifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_repor

# Model Initialization and Training
model = MLPClassifier(random_state=seed, max_iter=1000) # Increased max_iter for convergence
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Display the metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, pos_label='bad'))
print("Recall:", recall_score(y_test, y_pred, pos_label='bad'))
print("F1 Score:", f1_score(y_test, y_pred, pos_label='bad'))

class_names = ['good', 'bad']
print(classification_report(y_test, y_pred, target_names=class_names))
```

```
Accuracy: 0.675
Precision: 0.6376811594202898
Recall: 0.7586206896551724
F1 Score: 0.6929133858267716
              precision    recall  f1-score   support

        good       0.64      0.76      0.69        58
         bad       0.73      0.60      0.65        62

    accuracy                           0.68       120
   macro avg       0.68      0.68      0.67       120
weighted avg       0.68      0.68      0.67       120
```

```
# Train a GradientBoostingClassifier classifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

# Model Initialization and Training
model = GradientBoostingClassifier(random_state=seed)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]  # Get probability of the positive class ('bad')

# Display the metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, pos_label='bad'))
print("Recall:", recall_score(y_test, y_pred, pos_label='bad'))
print("F1 Score:", f1_score(y_test, y_pred, pos_label='bad'))
print("AUC:", roc_auc_score(y_test, y_pred_proba))

class_names = ['good', 'bad']
print(classification_report(y_test, y_pred, target_names=class_names))
```

```
Accuracy: 0.6416666666666667
Precision: 0.6056338028169014
Recall: 0.7413793103448276
F1 Score: 0.6666666666666666
AUC: 0.6927141268075638
              precision    recall  f1-score   support

        good       0.61      0.74      0.67        58
         bad       0.69      0.55      0.61        62

    accuracy                           0.64       120
   macro avg       0.65      0.64      0.64       120
weighted avg       0.65      0.64      0.64       120
```