Data Preprocessing

Цель этапа

Цель этапа **Data Preprocessing** — подготовить данные таким образом, чтобы каждая модель машинного обучения получила **оптимальное представление признаков** с учётом своей природы, допущений и ограничений.

Так как разные модели по-разному работают с масштабом признаков, категориальными данными и нелинейностями, preprocessing выполняется **индивидуально для каждой модели**.

Все решения на этом этапе основаны на выводах Exploratory Data Analysis (EDA).

Загрузка датасета

In [26]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
import joblib
```

In [27]:
```
columns = [
    "checking_account",
    "duration_months",
    "credit_history",
    "purpose",
    "credit_amount",
    "savings_account",
    "employment_duration",
    "installment_rate",
    "personal_status",
    "other_debtors",
    "present_residence",
    "property",
    "age",
    "other_installment_plans",
    "housing",
    "existing_credits",
    "job",
    "num_dependents",
    "telephone",
    "foreign_worker",
    "credit_risk"
]
df= pd.read_csv(
    "C:\\Users\\nurs\\OneDrive\\Рабочий стол\\german.data",
    sep=" ",
    header=None,
    names=columns
```

```
)

df.head()
```

Out[27]:

	checking_account	duration_months	credit_history	purpose	credit_amount	savings_
0	A11	6	A34	A43	1169	
1	A12	48	A32	A43	5951	
2	A14	12	A34	A46	2096	
3	A11	42	A32	A42	7882	
4	A11	24	A33	A40	4870	

5 rows × 21 columns

In [28]:

```
# ================================
# 2. TARGET TRANSFORMATION
# 1 = good, 2 = bad  →  1 = good, 0 = bad
# ================================

df["credit_risk"] = df["credit_risk"].map({1: 1, 2: 0})

# ================================
# 3. FULL HUMAN-READABLE MAPPING
# ================================

df["checking_account"] = df["checking_account"].map({
    "A11": "< 0 DM",
    "A12": "0–200 DM",
    "A13": ">= 200 DM",
    "A14": "no checking account"
})

df["credit_history"] = df["credit_history"].map({
    "A30": "no credits / all paid",
    "A31": "all credits paid back",
    "A32": "existing credits paid",
    "A33": "delay in the past",
    "A34": "critical account"
})

df["purpose"] = df["purpose"].map({
    "A40": "car (new)",
    "A41": "car (used)",
    "A42": "furniture/equipment",
    "A43": "radio/TV",
    "A44": "domestic appliances",
    "A45": "repairs",
    "A46": "education",
    "A47": "vacation",
    "A48": "retraining",
    "A49": "business",
    "A410": "other"
})

df["savings_account"] = df["savings_account"].map({
```

```
    "A61": "< 100 DM",
    "A62": "100-500 DM",
    "A63": "500-1000 DM",
    "A64": ">= 1000 DM",
    "A65": "unknown / none"
})

df["employment_duration"] = df["employment_duration"].map({
    "A71": "unemployed",
    "A72": "< 1 year",
    "A73": "1-4 years",
    "A74": "4-7 years",
    "A75": ">= 7 years"
})

df["personal_status"] = df["personal_status"].map({
    "A91": "male divorced/separated",
    "A92": "female divorced/separated/married",
    "A93": "male single",
    "A94": "male married/widowed",
    "A95": "female single"
})

df["other_debtors"] = df["other_debtors"].map({
    "A101": "none",
    "A102": "co-applicant",
    "A103": "guarantor"
})

df["property"] = df["property"].map({
    "A121": "real estate",
    "A122": "life insurance / savings",
    "A123": "car / other",
    "A124": "unknown / none"
})

df["other_installment_plans"] = df["other_installment_plans"].map({
    "A141": "bank",
    "A142": "stores",
    "A143": "none"
})

df["housing"] = df["housing"].map({
    "A151": "rent",
    "A152": "own",
    "A153": "for free"
})

df["job"] = df["job"].map({
    "A171": "unemployed / unskilled (non-resident)",
    "A172": "unskilled (resident)",
    "A173": "skilled employee",
    "A174": "management / highly qualified"
})

df["telephone"] = df["telephone"].map({
    "A191": "no",
    "A192": "yes"
})
```

```
df["foreign_worker"] = df["foreign_worker"].map({
    "A201": "yes",
    "A202": "no"
})

# ==============================
# 4. FINAL CHECK
# ==============================

print(df.head())
```

```
     checking_account  duration_months         credit_history  \
0              < 0 DM                6       critical account
1            0–200 DM               48  existing credits paid
2  no checking account               12       critical account
3              < 0 DM               42  existing credits paid
4              < 0 DM               24      delay in the past

              purpose  credit_amount savings_account employment_duration  \
0            radio/TV           1169  unknown / none          >= 7 years
1            radio/TV           5951        < 100 DM           1–4 years
2           education           2096        < 100 DM           4–7 years
3  furniture/equipment           7882        < 100 DM           4–7 years
4           car (new)            4870        < 100 DM           1–4 years

   installment_rate                   personal_status other_debtors  ...  \
0                 4                       male single          none  ...
1                 2  female divorced/separated/married          none  ...
2                 2                       male single          none  ...
3                 2                       male single     guarantor  ...
4                 3                       male single          none  ...

                  property age  other_installment_plans   housing  \
0              real estate  67                     none       own
1              real estate  22                     none       own
2              real estate  49                     none       own
3  life insurance / savings  45                     none  for free
4           unknown / none  53                     none  for free

  existing_credits                  job num_dependents  telephone  \
0                2     skilled employee              1        yes
1                1     skilled employee              1         no
2                1  unskilled (resident)             2         no
3                1     skilled employee              2         no
4                2     skilled employee              2         no

  foreign_worker credit_risk
0            yes           1
1            yes           0
2            yes           1
3            yes           1
4            yes           0

[5 rows x 21 columns]
```

1. Разделение данных

Перед выполнением любой предобработки данные были разделены на обучающую и тестовую выборки.

- Используется `train_test_split` с параметром `stratify`
- Доля тестовой выборки — **20%**
- Сохранено распределение целевого класса (70% good / 30% bad)

Почему это важно:
Кредитный скоринг — задача бинарной классификации с умеренным дисбалансом классов. Стратифицированное разделение гарантирует корректную финальную оценку качества моделей.

In [29]:
```
# Target
target_col = "credit_risk"

# Feature groups (как в EDA)
num_cols = [
    "duration_months",
    "credit_amount",
    "installment_rate",
    "present_residence",
    "age",
    "existing_credits",
    "num_dependents"
]

cat_cols = [
    "checking_account",
    "credit_history",
    "purpose",
    "savings_account",
    "employment_duration",
    "personal_status",
    "other_debtors",
    "property",
    "other_installment_plans",
    "housing",
    "job",
    "telephone",
    "foreign_worker"
]

# X / y
X = df[num_cols + cat_cols].copy()
y = df[target_col].copy()

# Stratified split (важно из-за дисбаланса 70/30)
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Train shape:", X_train.shape, "Test shape:", X_test.shape)
print("Train target share:\n", y_train.value_counts(normalize=True).round(3))
print("Test target share:\n", y_test.value_counts(normalize=True).round(3))
```

```
Train shape: (800, 20) Test shape: (200, 20)
Train target share:
 credit_risk
1    0.7
0    0.3
Name: proportion, dtype: float64
Test target share:
 credit_risk
1    0.7
0    0.3
Name: proportion, dtype: float64
```

2. Разделение признаков

На основе EDA признаки были разделены на две группы:

Числовые признаки

- `duration_months`
- `credit_amount`
- `installment_rate`
- `present_residence`
- `age`
- `existing_credits`
- `num_dependents`

Категориальные признаки

- `checking_account`
- `credit_history`
- `purpose`
- `savings_account`
- `employment_duration`
- `personal_status`
- `other_debtors`
- `property`
- `other_installment_plans`
- `housing`
- `job`
- `telephone`
- `foreign_worker`

In [30]:
```
target_col = "credit_risk"
num_cols = [
    "duration_months",
    "credit_amount",
    "installment_rate",
    "present_residence",
    "age",
    "existing_credits",
```

```
    "num_dependents"
]

cat_cols = [
    "checking_account",
    "credit_history",
    "purpose",
    "savings_account",
    "employment_duration",
    "personal_status",
    "other_debtors",
    "property",
    "other_installment_plans",
    "housing",
    "job",
    "telephone",
    "foreign_worker"
]
X = df[num_cols + cat_cols].copy()
y = df[target_col].copy()
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Train shape:", X_train.shape, "Test shape:", X_test.shape)
print("Train target share:\n", y_train.value_counts(normalize=True).round(3))
print("Test target share:\n", y_test.value_counts(normalize=True).round(3))
```

```
Train shape: (800, 20) Test shape: (200, 20)
Train target share:
 credit_risk
1    0.7
0    0.3
Name: proportion, dtype: float64
Test target share:
 credit_risk
1    0.7
0    0.3
Name: proportion, dtype: float64
```

3. Feature Engineering

На основе выводов EDA были добавлены доменно-обоснованные признаки, отражающие финансовую нагрузку и социальную стабильность клиента.

Добавленные признаки

- **`credit_load`** — кредитная нагрузка в месяц `credit_amount / duration_months`
- **`is_young`** — молодой заёмщик (`age < 25`)
- **`has_savings`** — наличие сбережений (`savings_account != "< 100 DM"`)

- `stable_job` — стабильный стаж работы (`employment_duration ∈ {"4–7 years", ">= 7 years"}`)

Почему:
EDA показал, что высокая финансовая нагрузка, молодой возраст и нестабильная занятость связаны с повышенным кредитным риском.

Feature Engineering выполняется **одинаково для train и test**, без использования целевой переменной, что исключает утечку данных.

In [31]:
```
def add_features(df):
    df = df.copy()
    df["credit_load"] = df["credit_amount"] / df["duration_months"]
    df["is_young"] = (df["age"] < 25).astype(int)
    df["has_savings"] = (df["savings_account"] != "< 100 DM").astype(int)
    df["stable_job"] = df["employment_duration"].isin(
        ["4–7 years", ">= 7 years"]
    ).astype(int)
    return df
X_train_fe = add_features(X_train)
X_test_fe = add_features(X_test)
print("New train shape:", X_train_fe.shape)
```

```
New train shape: (800, 24)
```

4. Preprocessing для Logistic Regression

Logistic Regression чувствительна к масштабу признаков и мультиколлинеарности.

Используемые шаги

- **StandardScaler** для числовых признаков
- **One-Hot Encoding** для категориальных (`drop="first"`)
- `class_weight="balanced"` учитывается на этапе обучения

In [32]:
```
num_cols_lr = [
    "duration_months",
    "credit_amount",
    "installment_rate",
    "present_residence",
    "age",
    "existing_credits",
    "num_dependents",
    "credit_load",
    "is_young",
    "has_savings",
    "stable_job"
]
```

In [33]:
```
cat_cols_lr = cat_cols.copy()
```

In [34]:
```
lr_preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols_lr),
```

```
        (
            "cat",
            OneHotEncoder(
                drop="first",
                handle_unknown="ignore"
            ),
            cat_cols_lr
        )
    ]
)
```

In [35]:
```
lr_pipeline = Pipeline(
    steps=[
        ("preprocessor", lr_preprocessor),
        (
            "model",
            LogisticRegression(
                max_iter=3000,
                class_weight="balanced",
                solver="liblinear",
                random_state=42
            )
        )
    ]
)
```

In [36]:
```
X_lr_train = lr_pipeline["preprocessor"].fit_transform(X_train_fe)
X_lr_test = lr_pipeline["preprocessor"].transform(X_test_fe)
print("LR train shape:", X_lr_train.shape)
print("LR test shape:", X_lr_test.shape)
```

```
LR train shape: (800, 52)
LR test shape: (200, 52)
```

Итог

- Количество признаков после preprocessing: **52**
- Используется `ColumnTransformer` + `Pipeline`
- Данные готовы к baseline-обучению и регуляризации

5. Preprocessing для Random Forest

Random Forest не чувствителен к масштабу признаков, но не может работать с текстовыми данными.

Используемые шаги

- Числовые признаки передаются **без масштабирования**
- Категориальные признаки кодируются с помощью **One-Hot Encoding**
- `drop` не используется

In [37]:
```
num_cols_rf = [
    "duration_months",
    "credit_amount",
```

```
    "installment_rate",
    "present_residence",
    "age",
    "existing_credits",
    "num_dependents",
    "credit_load",
    "is_young",
    "has_savings",
    "stable_job"
]
cat_cols_rf = cat_cols.copy()
```

In [38]:
```
rf_preprocessor = ColumnTransformer(
    transformers=[
        ("num", "passthrough", num_cols_rf),
        (
            "cat",
            OneHotEncoder(
                drop=None,
                handle_unknown="ignore"
            ),
            cat_cols_rf
        )
    ]
)
```

In [39]:
```
X_rf_train = rf_preprocessor.fit_transform(X_train_fe)
X_rf_test = rf_preprocessor.transform(X_test_fe)
print("RF train shape:", X_rf_train.shape)
print("RF test shape:", X_rf_test.shape)
```

```
RF train shape: (800, 65)
RF test shape: (200, 65)
```

Итог

- Количество признаков после preprocessing: **65**
- Используется `ColumnTransformer`
- Данные готовы к обучению ансамблевой модели

6. Preprocessing для Gradient Boosting (CatBoost)

Используемые шаги

- Числовые и категориальные признаки передаются **в исходном виде**
- Категориальные признаки передаются по индексам
- Используются все engineered-фичи

In [40]:
```
catboost_features = (
    num_cols_rf + cat_cols_rf
)
X_cb_train = X_train_fe[catboost_features].copy()
X_cb_test = X_test_fe[catboost_features].copy()
```

In [41]:
```
cat_feature_indices = [
    X_cb_train.columns.get_loc(col)
    for col in cat_cols_rf
]
```

In [42]:
```
cat_feature_indices
```

Out[42]:
```
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
```

In [43]:
```
print("CatBoost train shape:", X_cb_train.shape)
print("CatBoost test shape:", X_cb_test.shape)
print("Number of categorical features:", len(cat_feature_indices))
```

```
CatBoost train shape: (800, 24)
CatBoost test shape: (200, 24)
Number of categorical features: 13
```

Итог

- Количество признаков: **24**
- Количество категориальных признаков: **13**
- Минимальная предобработка без потери информации

7. Сохранение preprocessing-артефактов

Для обеспечения воспроизводимости и чистой архитектуры проекта все preprocessing-объекты и данные были сохранены.

Сохранены:

- данные train/test
- preprocessing-объекты
- списки признаков
- индексы категориальных признаков для CatBoost

In [44]:
```
joblib.dump(
    {
        "X_train": X_train_fe,
        "X_test": X_test_fe,
        "y_train": y_train,
        "y_test": y_test,
        "preprocessor": lr_preprocessor,
        "num_cols": num_cols_lr,
        "cat_cols": cat_cols_lr
    },
    "lr_preprocessing.joblib"
)
```

Out[44]:
```
['lr_preprocessing.joblib']
```

In [45]:
```
joblib.dump(
    {
        "X_train": X_train_fe,
        "X_test": X_test_fe,
```

```
        "y_train": y_train,
        "y_test": y_test,
        "preprocessor": rf_preprocessor,
        "num_cols": num_cols_rf,
        "cat_cols": cat_cols_rf
    },
    "rf_preprocessing.joblib"
)
```

Out[45]: ['rf_preprocessing.joblib']

In [46]:
```
joblib.dump(
    {
        "X_train": X_cb_train,
        "X_test": X_cb_test,
        "y_train": y_train,
        "y_test": y_test,
        "cat_feature_indices": cat_feature_indices,
        "feature_names": catboost_features
    },
    "catboost_preprocessing.joblib"
)
```

Out[46]: ['catboost_preprocessing.joblib']

In [47]:
```
test_load = joblib.load("catboost_preprocessing.joblib")
print(test_load["X_train"].shape)
print(len(test_load["cat_feature_indices"]))
```

```
(800, 24)
13
```

Вывод

На этапе Data Preprocessing была реализована модельно-ориентированная стратегия подготовки данных, основанная на результатах EDA.

Каждая модель получает оптимальное представление признаков, что обеспечивает:

- корректное сравнение моделей
- высокую воспроизводимость
- соответствие лучшим практикам кредитного скоринга

Данные полностью готовы к этапу **Baseline Model Training**.