02_Data_preprocessing

March 20, 2025

1 MLOps Project work: Data preprocessing

2 MLOps Project work: Data preparation

2.1 1. Introduction

In this section, the preprocessing of the adult income data set wil be demonstrated. The focus will be on data cleaning and feature engineering.

2.2 2. Learning objectives

The following objectives will be showcasted: - Master methods for systematic data cleaning - Be able to apply feature engineering techniques - Be able to create a complete preprocessing pipeline - Understand preprocessing best practices in the MLOps context

2.3 3. Setup and data preparation

[1]:
```
# Benötigte Bibliotheken importieren
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
import pickle
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer


# Matplotlib für deutsche Beschriftungen konfigurieren
plt.rcParams['axes.formatter.use_locale'] = True
```

[2]:
```
# Datensatz laden
df = pd.read_csv('../data/raw/adult-income.csv')
print(f"Datensatz geladen: {df.shape} (Zeilen, Spalten)")
df
```

```
Datensatz geladen: (48842, 15) (Zeilen, Spalten)
```

[2]:
```
         age     workclass  fnlwgt     education  educational-num  \
0        25       Private  226802          11th                7
1        38       Private   89814       HS-grad                9
2        28     Local-gov  336951    Assoc-acdm               12
3        44       Private  160323  Some-college               10
4        18             ?  103497  Some-college               10
…     …           …      …           …               …
48837    27       Private  257302    Assoc-acdm               12
48838    40       Private  154374       HS-grad                9
48839    58       Private  151910       HS-grad                9
48840    22       Private  201490       HS-grad                9
48841    52  Self-emp-inc  287927       HS-grad                9

           marital-status         occupation relationship   race  gender  \
0           Never-married  Machine-op-inspct    Own-child  Black    Male
1      Married-civ-spouse    Farming-fishing      Husband  White    Male
2      Married-civ-spouse    Protective-serv      Husband  White    Male
3      Married-civ-spouse  Machine-op-inspct      Husband  Black    Male
4           Never-married                  ?    Own-child  White  Female
…                     …                  …          …     …     …
48837  Married-civ-spouse       Tech-support         Wife  White  Female
48838  Married-civ-spouse  Machine-op-inspct      Husband  White    Male
48839             Widowed       Adm-clerical    Unmarried  White  Female
48840       Never-married       Adm-clerical    Own-child  White    Male
48841  Married-civ-spouse    Exec-managerial         Wife  White  Female

       capital-gain  capital-loss  hours-per-week native-country income
0                 0             0              40  United-States  <=50K
1                 0             0              50  United-States  <=50K
2                 0             0              40  United-States   >50K
3              7688             0              40  United-States   >50K
4                 0             0              30  United-States  <=50K
…               …             …              …            …    …
48837             0             0              38  United-States  <=50K
48838             0             0              40  United-States   >50K
48839             0             0              40  United-States  <=50K
48840             0             0              20  United-States  <=50K
48841         15024             0              40  United-States   >50K

[48842 rows x 15 columns]
```

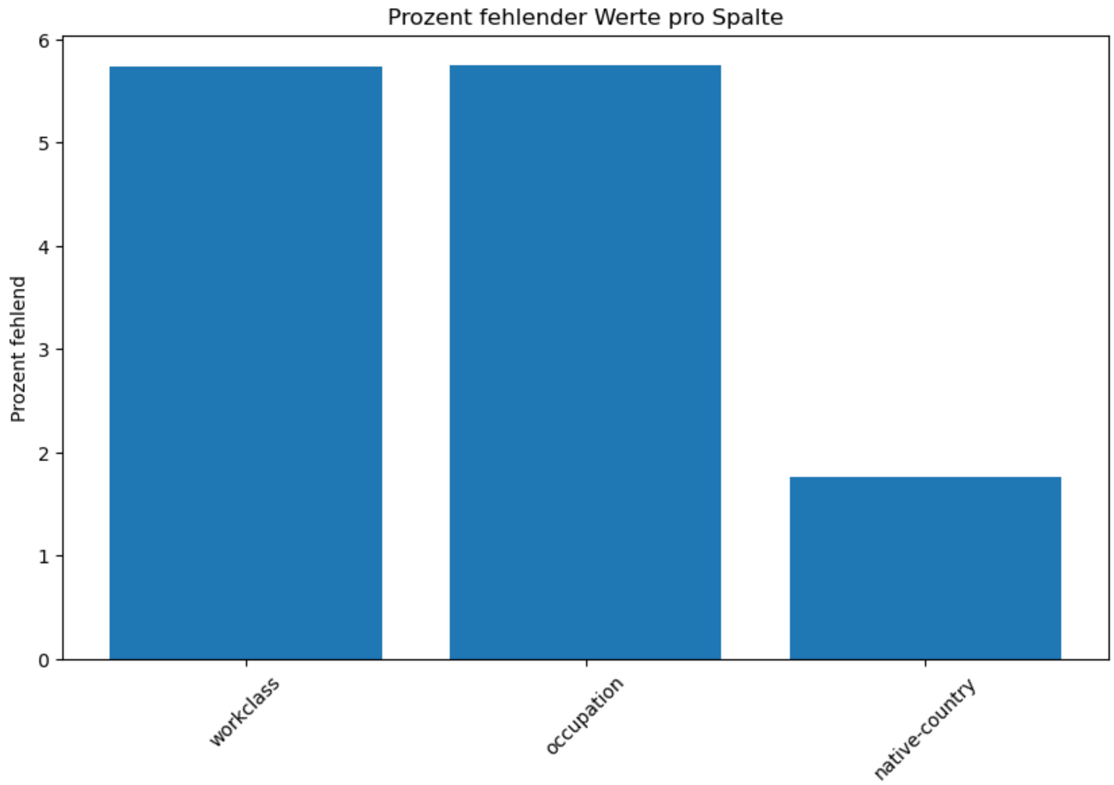
2.4 4. Data cleaning

2.4.1 4.1 Handle missing values

Firstly, the missing values are analyzed. The data set is examined for missing values. Secondly, the distribution of missing values will be visualized. Lastly, a strategy to handle missing values will be developed and implemented.

We will first check for missing values. From the previous exploratory data anaylsis, we know that the missing values are marked with '?' in the dataset.

[3]:
```
# missing values are marked as '?' in this dataset

def analyse_fehlende_werte(df):
    # number of missing values oer column
    fehlend = (df == '?').sum()

    # Percentage of missing values
    fehlend_prozent = (fehlend / len(df)) * 100

    # DataFrame with results
    fehlend_info = pd.DataFrame({
        'Fehlende Werte': fehlend,
        'Prozent Fehlend': fehlend_prozent
    })

    # visualizing
    plt.figure(figsize=(10, 6))
    plt.bar(fehlend_info[fehlend_info['Fehlende Werte'] > 0].index,
            fehlend_info[fehlend_info['Fehlende Werte'] > 0]['Prozent Fehlend'])
    plt.title('Prozent fehlender Werte pro Spalte')
    plt.xticks(rotation=45)
    plt.ylabel('Prozent fehlend')
    plt.show()

    return fehlend_info[fehlend_info['Fehlende Werte'] > 0]


# perform analysis
fehlend_analyse = analyse_fehlende_werte(df)
print("Analyse fehlender Werte:")
print(fehlend_analyse)
```




```
Analyse fehlender Werte:
                Fehlende Werte  Prozent Fehlend
workclass                 2799         5.730724
occupation                2809         5.751198
native-country             857         1.754637
```

The analysis confirms that our missing values are '?' in the dataset. Data are missing in the columns 'workclass', 'occupation', 'native-country'.

[6]:
```
# check for '?' in the dataset

df.isin(['?']).sum()
```

[6]:
```
age                0
workclass          0
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race               0
gender             0
```

```
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
```

We can observe that Workclass has 2799 missing values, occupation has 2809 missing values and country has 857 missing values. The rows with '?' are to be replaced with NaN for better data handling.

[5]:
```
# replace '?' with Unknown

df['workclass']=df['workclass'].replace('?','Unknown')
df['occupation']=df['occupation'].replace('?','Unknown')
df['native-country']=df['native-country'].replace('?','Unknown')
df.head()
```

[5]:
```
   age  workclass  fnlwgt     education  educational-num      marital-status  \
0   25    Private  226802          11th                7        Never-married
1   38    Private   89814       HS-grad                9  Married-civ-spouse
2   28  Local-gov  336951    Assoc-acdm               12  Married-civ-spouse
3   44    Private  160323  Some-college               10  Married-civ-spouse
4   18    Unknown  103497  Some-college               10        Never-married

          occupation relationship   race  gender  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male             0             0
1    Farming-fishing      Husband  White    Male             0             0
2    Protective-serv      Husband  White    Male             0             0
3  Machine-op-inspct      Husband  Black    Male          7688             0
4            Unknown    Own-child  White  Female             0             0

   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K
```

After replacing '?' with "Unknown" in the dataset, we then proceed with creating an overview about the dataset. The overviews consists information about IsNull, IsNa, Duplicate, Unique, Min, Max.

[7]:
```
# check for other missing values

info = pd.DataFrame(df.isnull().sum(),columns=["IsNull"])
info.insert(1,"IsNa",df.isna().sum(),True)
info.insert(2,"Duplicate",df.duplicated().sum(),True)
```

```python
info.insert(3,"Unique",df.nunique(),True)


# min and max is not applied to string value
#info.insert(4,"Min",data.min(),True)
#info.insert(5,"Max",data.max(),True)


numeric_cols = ['age', 'fnlwgt', 'educational-num', 'capital-gain',
 ↪'capital-loss', 'hours-per-week']
numeric_data = df[numeric_cols]  # Create a DataFrame with only the numeric
 ↪columns

if not numeric_data.empty:
    info.insert(4, "Min", numeric_data.min(), True)
    info.insert(5, "Max", numeric_data.max(), True)
else:
    print("No numeric columns found in the dataframe.")

info.T
```

[7]:

```
            age  workclass     fnlwgt  education  educational-num  \
IsNull      0.0        0.0        0.0        0.0              0.0
IsNa        0.0        0.0        0.0        0.0              0.0
Duplicate  52.0       52.0       52.0       52.0             52.0
Unique     74.0        9.0    28523.0       16.0             16.0
Min        17.0        NaN    12285.0        NaN              1.0
Max        90.0        NaN  1490400.0        NaN             16.0

           marital-status  occupation  relationship  race  gender  \
IsNull                0.0         0.0           0.0   0.0     0.0
IsNa                  0.0         0.0           0.0   0.0     0.0
Duplicate            52.0        52.0          52.0  52.0    52.0
Unique                7.0        15.0           6.0   5.0     2.0
Min                   NaN         NaN           NaN   NaN     NaN
Max                   NaN         NaN           NaN   NaN     NaN

           capital-gain  capital-loss  hours-per-week  native-country  income
IsNull              0.0           0.0             0.0             0.0     0.0
IsNa                0.0           0.0             0.0             0.0     0.0
Duplicate          52.0          52.0            52.0            52.0    52.0
Unique            123.0          99.0            96.0            42.0     2.0
Min                 0.0           0.0             1.0             NaN     NaN
Max             99999.0        4356.0            99.0             NaN     NaN
```

We are going to replace the missing values in the dataset with the following strategy:

- median for numerical colums: numerical missing values are replaced by median value of the

data set
- most frequent for categorical colums: categorical missing values are replaced by most-frequent value of the column

To sum up, the dataset was not using the default NaN string for missing values, instead "?" was used. The null or NaN values were marked as '?'. The three columns 'workclass','occupation','native-country have'?' values.

Based on the results of data exploration, we also know that there are also duplicates in the dataset. 52 duplicate rows need to be further handled. We proceed with removing duplicated from the dataset.

2.4.2 4.2 Handle duplicates and missing values

Furthermore, we discovered that duplicated columns convey the same information such as "education" and "educational-num". Both are in fact same information. Furthermore, some columns have very detailed classification grade, for example race. The occupation is further classified into different workclasses. For example, machine-op-inspection, farming-fishing, craft-repair etc are classified into private workclass.

In this section, duplicates are identified and cleaned up. We will start with checking the record for different types of duplicates, analyzing found duplicates and deciding how to handle duplicates. At last, duplicates are cleaned up.

[9]:
```
# remove duplicates

df = df.drop_duplicates()
df.shape
df
```

[9]:
```
        age     workclass    fnlwgt     education  educational-num  \
0      25.0       Private  226802.0          11th              7.0
1      38.0       Private   89814.0       HS-grad              9.0
2      28.0     Local-gov  336951.0    Assoc-acdm             12.0
3      44.0       Private  160323.0  Some-college             10.0
4      18.0       Unknown  103497.0  Some-college             10.0
…      …              …         …             …                …
48837  27.0       Private  257302.0    Assoc-acdm             12.0
48838  40.0       Private  154374.0       HS-grad              9.0
48839  58.0       Private  151910.0       HS-grad              9.0
48840  22.0       Private  201490.0       HS-grad              9.0
48841  52.0  Self-emp-inc  287927.0       HS-grad              9.0

           marital-status         occupation relationship   race  gender  \
0           Never-married  Machine-op-inspct    Own-child  Black    Male
1      Married-civ-spouse    Farming-fishing      Husband  White    Male
2      Married-civ-spouse    Protective-serv      Husband  White    Male
3      Married-civ-spouse  Machine-op-inspct      Husband  Black    Male
4           Never-married            Unknown    Own-child  White  Female
```

```
…                      …                  …         …      …
48837  Married-civ-spouse       Tech-support      Wife  White  Female
48838  Married-civ-spouse  Machine-op-inspct   Husband  White    Male
48839             Widowed       Adm-clerical  Unmarried  White  Female
48840       Never-married       Adm-clerical  Own-child  White    Male
48841  Married-civ-spouse    Exec-managerial      Wife  White  Female

       capital-gain  capital-loss  hours-per-week native-country income
0               0.0           0.0            40.0  United-States  <=50K
1               0.0           0.0            50.0  United-States  <=50K
2               0.0           0.0            40.0  United-States   >50K
3            7688.0           0.0            40.0  United-States   >50K
4               0.0           0.0            30.0  United-States  <=50K
…               …             …               …             …     …
48837           0.0           0.0            38.0  United-States  <=50K
48838           0.0           0.0            40.0  United-States   >50K
48839           0.0           0.0            40.0  United-States  <=50K
48840           0.0           0.0            20.0  United-States  <=50K
48841       15024.0           0.0            40.0  United-States   >50K

[48790 rows x 15 columns]
```

We proceeed with removing duplicates. The dataset is cleansed from duplicates. After removing duplicates, we have 48790 rows (with 15 columns) in the dataset.

"fnlwgt" is dropped as it represents census weights that are not relevant to our project's goal

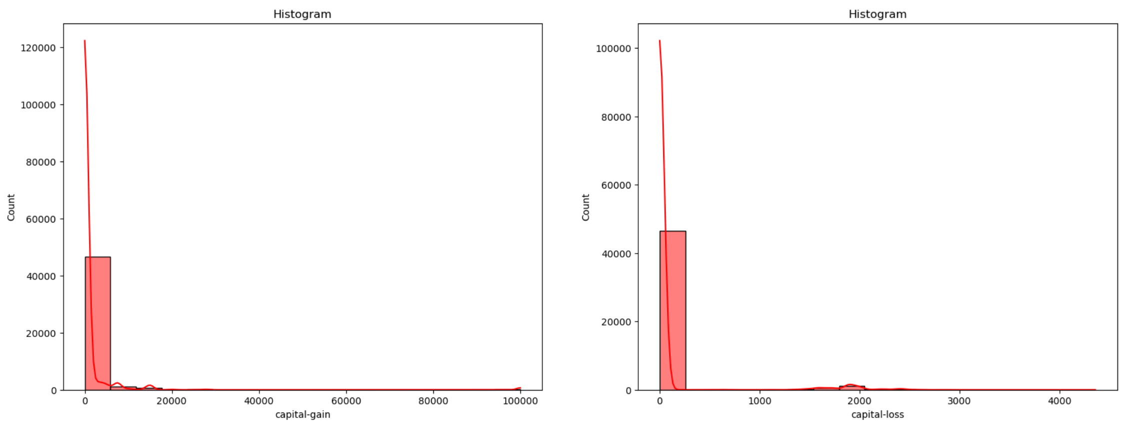
[10]:
```
df=df.drop(columns=["fnlwgt"])
df.head()
```

[10]:
```
    age  workclass     education  educational-num      marital-status  \
0  25.0    Private          11th              7.0       Never-married
1  38.0    Private       HS-grad              9.0  Married-civ-spouse
2  28.0  Local-gov    Assoc-acdm             12.0  Married-civ-spouse
3  44.0    Private  Some-college             10.0  Married-civ-spouse
4  18.0    Unknown  Some-college             10.0       Never-married

          occupation relationship   race  gender  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male           0.0           0.0
1    Farming-fishing      Husband  White    Male           0.0           0.0
2    Protective-serv      Husband  White    Male           0.0           0.0
3  Machine-op-inspct      Husband  Black    Male        7688.0           0.0
4            Unknown    Own-child  White  Female           0.0           0.0

  hours-per-week native-country income
0           40.0  United-States  <=50K
1           50.0  United-States  <=50K
2           40.0  United-States   >50K
```

```
3             40.0  United-States   >50K
4             30.0  United-States  <=50K
```

[197]:
```
# plot histogram for capital-gain and capital-loss


plt.figure(figsize=(20, 7))
plt.subplot(1, 2, 1)
sns.histplot(df['capital-gain'], kde = True,color='r')
plt.title('Histogram')
plt.subplot(1, 2, 2)
sns.histplot(df['capital-loss'], kde = True,color='r')
plt.title('Histogram')
```

[197]:
```
Text(0.5, 1.0, 'Histogram')
```

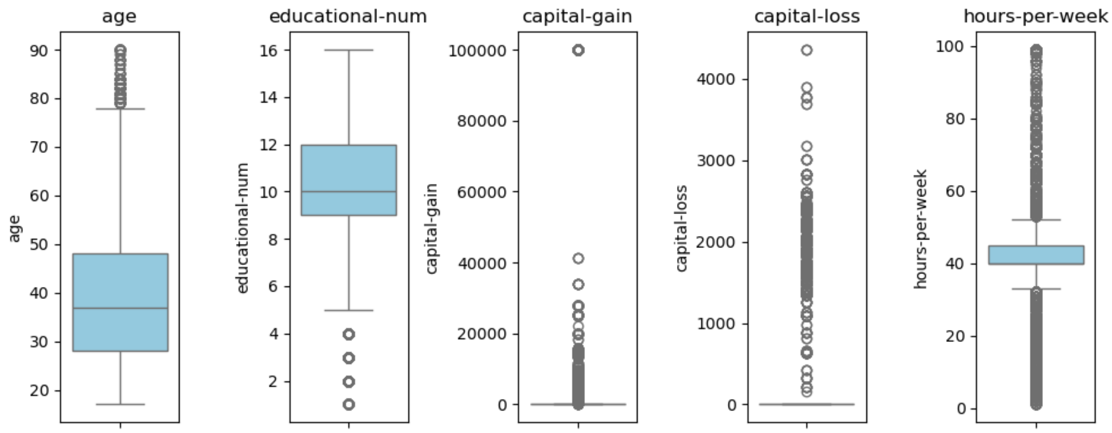


"capital-gain" and "capital-loss" both columns over 90% data as 0 (capital-gain 91% and capital-loss 95% with value 0).

Next, we check for outliers and remove them.

[11]:
```
# visualizing outliers
x = 0
#Numerical features;
numeric_columns = df.select_dtypes(include=['number']).columns

# Create a figure with specified size
plt.figure(figsize=(16, 4))

for col in numeric_columns:
    x += 1
    plt.subplot(1, 8, x)
    sns.boxplot(data=df[col], color='skyblue')
```

```
    plt.title(col)

plt.tight_layout()
plt.show()


# size of data frame before removing outliers
df.shape
```



[11]: (48790, 14)

We will remove outliers in the dataset.

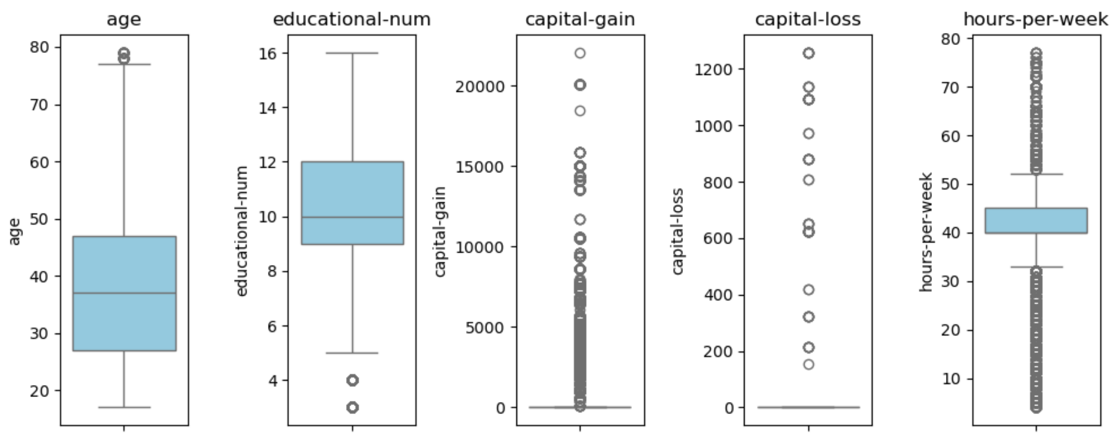
[12]:
```
import numpy as np
from scipy import stats


# function to remove outliers based on Z-Score
def remove_outliers_zscore(df, threshold=3):
    numeric_columns = df.select_dtypes(include=['number']).columns  # Wähle␣
 ↪numerische Spalten aus

    # Berechne den Z-Score für numerische Spalten
    z_scores = np.abs(stats.zscore(df[numeric_columns]))  # Absoluten Wert des␣
 ↪Z-Scores nehmen

    # Filtere Zeilen heraus, bei denen ein Wert einen Z-Score > threshold hat
    df_clean = df[(z_scores < threshold).all(axis=1)]

    return df_clean

# function to remove outliers
```

```
df = remove_outliers_zscore(df)

# show size of data frame after removing outliers
df.shape
```

[12]: (45139, 14)

Numeric columns without outliers are visualized as below.

[13]:
```
# Checking Outliers in Individual Features
x = 0
#Numerical features;
numeric_columns = df.select_dtypes(include=['number']).columns

# Create a figure with specified size
plt.figure(figsize=(16, 4))

for col in numeric_columns:
    x += 1
    plt.subplot(1, 8, x)
    sns.boxplot(data=df[col], color='skyblue')
    plt.title(col)

plt.tight_layout()
plt.show()

df.shape
```

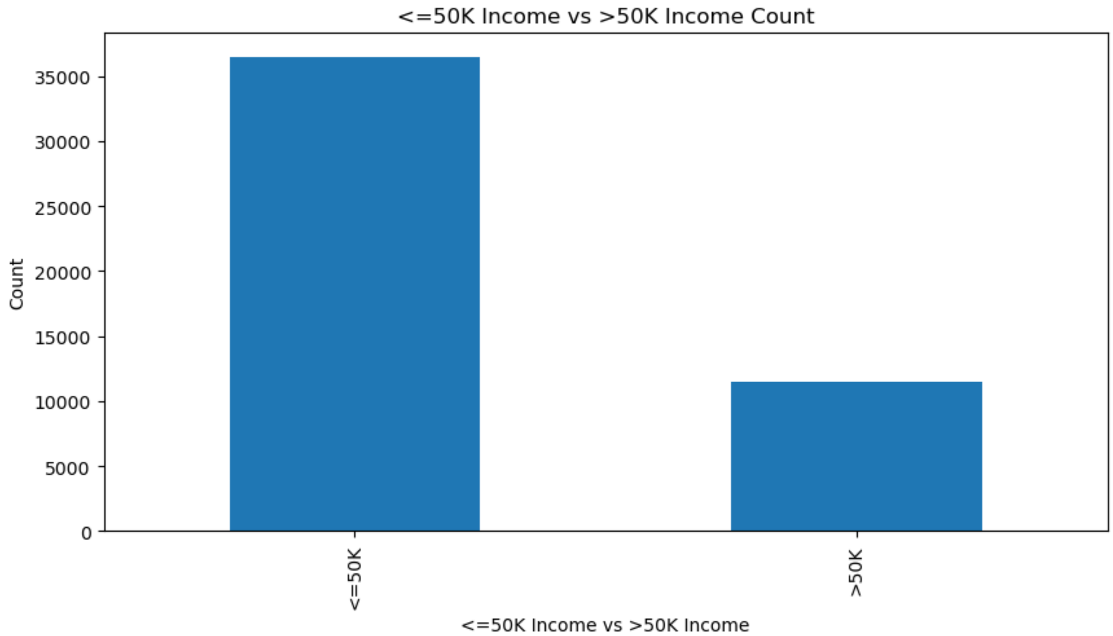


[13]: (45139, 14)

[202]:
```
# count of target variable 'income'
```

```
plt.figure(figsize=(10,5))
ax = df.income.value_counts().plot(kind = 'bar')
plt.xlabel("<=50K Income vs >50K Income")
plt.ylabel("Count")
plt.title("<=50K Income vs >50K Income Count")
```

[202]: Text(0.5, 1.0, '<=50K Income vs >50K Income Count')



2.5 5. Feature Engineering

2.5.1 5.1 Encoding categorical variables

We will prepare the categorical variables for the next step. All categorical variables in the dataset are identified, the cardinality of each categorical variable is analyzed. Next, appropriate encoding methods are choosen for different variable types and implemented.

[203]:
```
def kategorische_variablen_kodieren(df):
    # Kategorische Spalten identifizieren
    kategorische_spalten = df.select_dtypes(include=['object']).columns
    #kategorische_spalten = kategorische_spalten[kategorische_spalten!
 ↪="customerID"]
    # Kardinalität analysieren
    kardinalitaet = pd.DataFrame({
        'Spalte': kategorische_spalten,
        'Unique_Werte': [df[spalte].nunique() for spalte in␣
 ↪kategorische_spalten]
```

```
    })
    print("Kardinalität der kategorischen Variablen:")
    print(kardinalitaet)

    # perform encoding
    encoded_df = df.copy()

    # label encoding for binary variables
    binary_encoder = LabelEncoder()
    for spalte in kategorische_spalten:
        if df[spalte].nunique() == 2:
            encoded_df[spalte] = binary_encoder.fit_transform(df[spalte])
        else:
            # One-Hot Encoding für nicht-binäre Variablen
            dummies = pd.get_dummies(df[spalte], prefix=spalte)
            encoded_df = pd.concat([encoded_df, dummies], axis=1)
            encoded_df.drop(spalte, axis=1, inplace=True)

    return encoded_df

# perform encoding
encoded_df = kategorische_variablen_kodieren(df)
print("\nNeue Features nach Encoding:")
print(encoded_df.columns.tolist())
```

```
Kardinalität der kategorischen Variablen:
           Spalte  Unique_Werte
0       workclass             8
1       education            16
2  marital-status             7
3      occupation            14
4    relationship             6
5            race             5
6          gender             2
7  native-country            41
8          income             2

Neue Features nach Encoding:
['age', 'gender', 'hours-per-week', 'income', 'workclass_Federal-gov',
'workclass_Local-gov', 'workclass_Never-worked', 'workclass_Private',
'workclass_Self-emp-inc', 'workclass_Self-emp-not-inc', 'workclass_State-gov',
'workclass_Without-pay', 'education_10th', 'education_11th', 'education_12th',
'education_1st-4th', 'education_5th-6th', 'education_7th-8th', 'education_9th',
'education_Assoc-acdm', 'education_Assoc-voc', 'education_Bachelors',
'education_Doctorate', 'education_HS-grad', 'education_Masters',
'education_Preschool', 'education_Prof-school', 'education_Some-college',
'marital-status_Divorced', 'marital-status_Married-AF-spouse', 'marital-
status_Married-civ-spouse', 'marital-status_Married-spouse-absent', 'marital-
```

```
status_Never-married', 'marital-status_Separated', 'marital-status_Widowed',
'occupation_Adm-clerical', 'occupation_Armed-Forces', 'occupation_Craft-repair',
'occupation_Exec-managerial', 'occupation_Farming-fishing',
'occupation_Handlers-cleaners', 'occupation_Machine-op-inspct',
'occupation_Other-service', 'occupation_Priv-house-serv', 'occupation_Prof-
specialty', 'occupation_Protective-serv', 'occupation_Sales', 'occupation_Tech-
support', 'occupation_Transport-moving', 'relationship_Husband',
'relationship_Not-in-family', 'relationship_Other-relative', 'relationship_Own-
child', 'relationship_Unmarried', 'relationship_Wife', 'race_Amer-Indian-
Eskimo', 'race_Asian-Pac-Islander', 'race_Black', 'race_Other', 'race_White',
'native-country_Cambodia', 'native-country_Canada', 'native-country_China',
'native-country_Columbia', 'native-country_Cuba', 'native-country_Dominican-
Republic', 'native-country_Ecuador', 'native-country_El-Salvador', 'native-
country_England', 'native-country_France', 'native-country_Germany', 'native-
country_Greece', 'native-country_Guatemala', 'native-country_Haiti', 'native-
country_Holand-Netherlands', 'native-country_Honduras', 'native-country_Hong',
'native-country_Hungary', 'native-country_India', 'native-country_Iran',
'native-country_Ireland', 'native-country_Italy', 'native-country_Jamaica',
'native-country_Japan', 'native-country_Laos', 'native-country_Mexico', 'native-
country_Nicaragua', 'native-country_Outlying-US(Guam-USVI-etc)', 'native-
country_Peru', 'native-country_Philippines', 'native-country_Poland', 'native-
country_Portugal', 'native-country_Puerto-Rico', 'native-country_Scotland',
'native-country_South', 'native-country_Taiwan', 'native-country_Thailand',
'native-country_Trinadad&Tobago', 'native-country_United-States', 'native-
country_Vietnam', 'native-country_Yugoslavia']
```

2.5.2 5.2 Generating features

The dataset contains a "native-country" feature. Howerver, other than USA, many of the features have very low numbers of observations, so they are grouped into a single category.

[204]:
```python
# Replace non-'United-States' values in 'native-country' with 'Others'

df.loc[df["native-country"] != "United-States", "native-country"] = "Others"
df['native-country'].unique()
```

[204]:
```
array(['United-States', 'Others'], dtype=object)
```

In addition, the dataset has too much categories. These categories can be limited to make the dataset more clearer. We are going to limit the categorization of education and marital-status. They will limited as follows:

- education: dropout, HighGrad (high school graduate), CommunityCollege, Bachelors, Masters, Doctorate
- marital status: NotMarried, Married, Separated, Widowed
- race: White, Others

This reduces the educational levels from 16 to 6, marital status from 7 to 4 and race from 5 to 2.

[205]:
```python
# limit categorization of education
df['education'].replace('Preschool', 'dropout',inplace=True)
df['education'].replace('10th', 'dropout',inplace=True)
df['education'].replace('11th', 'dropout',inplace=True)
df['education'].replace('12th', 'dropout',inplace=True)
df['education'].replace('1st-4th', 'dropout',inplace=True)
df['education'].replace('5th-6th', 'dropout',inplace=True)
df['education'].replace('7th-8th', 'dropout',inplace=True)
df['education'].replace('9th', 'dropout',inplace=True)
df['education'].replace('HS-Grad', 'HighGrad',inplace=True)
df['education'].replace('HS-grad', 'HighGrad',inplace=True)
df['education'].replace('Some-college', 'CommunityCollege',inplace=True)
df['education'].replace('Assoc-acdm', 'CommunityCollege',inplace=True)
df['education'].replace('Assoc-voc', 'CommunityCollege',inplace=True)
df['education'].replace('Bachelors', 'Bachelors',inplace=True)
df['education'].replace('Masters', 'Masters',inplace=True)
df['education'].replace('Prof-school', 'Masters',inplace=True)
df['education'].replace('Doctorate', 'Doctorate',inplace=True)

df['education'].unique()
```

[205]:
```
array(['dropout', 'HighGrad', 'CommunityCollege', 'Masters', 'Bachelors',
       'Doctorate'], dtype=object)
```

[206]:
```python
# limit categorization of marital status
df['marital-status'].replace('Never-married', 'NotMarried',inplace=True)
df['marital-status'].replace(['Married-AF-spouse'], 'Married',inplace=True)
df['marital-status'].replace(['Married-civ-spouse'], 'Married',inplace=True)
df['marital-status'].replace(['Married-spouse-absent'],␣
↪'NotMarried',inplace=True)
df['marital-status'].replace(['Separated'], 'Separated',inplace=True)
df['marital-status'].replace(['Divorced'], 'Separated',inplace=True)
df['marital-status'].replace(['Widowed'], 'Widowed',inplace=True)

df['marital-status'].unique()
```

[206]:
```
array(['NotMarried', 'Married', 'Widowed', 'Separated'], dtype=object)
```

[207]:
```python
# limit categorization of race
df['race'].replace('Black', 'Others',inplace=True)
df['race'].replace(['Amer-Indian-Eskimo'], 'Others',inplace=True)
df['race'].replace(['Other'], 'Others',inplace=True)
df['race'].replace(['Asian-Pac-Islander'], 'Others',inplace=True)

df['race'].unique()
```

[207]:
```
array(['Others', 'White'], dtype=object)
```

[16]:
```python
# adding new features of age groups "Young", "Middle-aged", "Experienced",␣
↪"Senior"
# Define age bins and corresponding labels


bins = [0, 25, 40, 60, 100]
labels = ["Young", "Middle-aged", "Experienced", "Senior"]

# Create a new column "altersgruppe" (age group) based on age bins
df["altersgruppe"] = pd.cut(df["age"], bins=bins, labels=labels, right=False)

df.head()
```

[16]:
```
    age  workclass     education  educational-num      marital-status  \
0  25.0    Private          11th              7.0       Never-married
1  38.0    Private       HS-grad              9.0  Married-civ-spouse
2  28.0  Local-gov    Assoc-acdm             12.0  Married-civ-spouse
3  44.0    Private  Some-college             10.0  Married-civ-spouse
4  18.0    Unknown  Some-college             10.0       Never-married

          occupation relationship   race  gender  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male           0.0           0.0
1    Farming-fishing      Husband  White    Male           0.0           0.0
2    Protective-serv      Husband  White    Male           0.0           0.0
3  Machine-op-inspct      Husband  Black    Male        7688.0           0.0
4            Unknown    Own-child  White  Female           0.0           0.0

   hours-per-week native-country income altersgruppe
0            40.0  United-States  <=50K  Middle-aged
1            50.0  United-States  <=50K  Middle-aged
2            40.0  United-States   >50K  Middle-aged
3            40.0  United-States   >50K  Experienced
4            30.0  United-States  <=50K        Young
```

There were missing values in the dataset with placeholders '?'. Now we recheck the dataset, there are no more missing values exist in the dataset.

[19]:
```python
df.isin(['?']).sum()

# '?' have been replaced with 'Unknown'
```

[19]:
```
age                0
workclass          0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
```

```
race              0
gender            0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
income            0
altersgruppe      0
dtype: int64
```

[20]:

```
# check for other missing values

info = pd.DataFrame(df.isnull().sum(),columns=["IsNull"])
info.insert(1,"IsNa",df.isna().sum(),True)
info.insert(2,"Duplicate",df.duplicated().sum(),True)
info.insert(3,"Unique",df.nunique(),True)


# min and max is not applied to string value
#info.insert(4,"Min",data.min(),True)
#info.insert(5,"Max",data.max(),True)


numeric_cols = ['age', 'hours-per-week']
numeric_data = df[numeric_cols]  # Create a DataFrame with only the numeric␣
↪columns

if not numeric_data.empty:
    info.insert(4, "Min", numeric_data.min(), True)
    info.insert(5, "Max", numeric_data.max(), True)
else:
    print("No numeric columns found in the dataframe.")

info.T
```

[20]:

```
              age  workclass  education  educational-num  marital-status  \
IsNull        0.0        0.0        0.0              0.0             0.0
IsNa          0.0        0.0        0.0              0.0             0.0
Duplicate  6292.0     6292.0     6292.0           6292.0          6292.0
Unique       63.0        9.0       14.0             14.0             7.0
Min          17.0        NaN        NaN              NaN             NaN
Max          79.0        NaN        NaN              NaN             NaN

           occupation  relationship    race  gender  capital-gain  \
IsNull            0.0           0.0     0.0     0.0           0.0
IsNa              0.0           0.0     0.0     0.0           0.0
Duplicate      6292.0        6292.0  6292.0  6292.0        6292.0
```

```
Unique           15.0           6.0      5.0      2.0          117.0
Min               NaN           NaN      NaN      NaN            NaN
Max               NaN           NaN      NaN      NaN            NaN

            capital-loss  hours-per-week  native-country  income  altersgruppe
IsNull               0.0             0.0             0.0     0.0           0.0
IsNa                 0.0             0.0             0.0     0.0           0.0
Duplicate         6292.0          6292.0          6292.0  6292.0        6292.0
Unique              13.0            73.0            41.0     2.0           4.0
Min                  NaN             4.0             NaN     NaN           NaN
Max                  NaN            77.0             NaN     NaN           NaN
```

[21]:

```
# Number of rows that have one null values
one_null = sum(df['workclass'].isnull() & ~df['occupation'].isnull() &␣
 ↪~df['native-country'].isnull()) \
           + sum(~df['workclass'].isnull() & df['occupation'].isnull() &␣
 ↪~df['native-country'].isnull()) \
           + sum(~df['workclass'].isnull() & ~df['occupation'].isnull() &␣
 ↪df['native-country'].isnull())

# Number of rows that have two null values
two_null = sum(df['workclass'].isnull() & df['occupation'].isnull() &␣
 ↪~df['native-country'].isnull()) \
           + sum(df['workclass'].isnull() & ~df['occupation'].isnull() &␣
 ↪df['native-country'].isnull()) \
           + sum(~df['workclass'].isnull() & df['occupation'].isnull() &␣
 ↪df['native-country'].isnull())

# Number of rows that have three null values
three_null = sum(df['workclass'].isnull() & df['occupation'].isnull() &␣
 ↪df['native-country'].isnull())

# Print the number of rows that have one, two and three null values
print('Number of rows that have one null values:', one_null)
print('Number of rows that have two null values:', two_null)
print('Number of rows that have three null values:', three_null)
```

```
Number of rows that have one null values: 0
Number of rows that have two null values: 0
Number of rows that have three null values: 0
```

[22]:

```
# before label encoding
df.head()
```

[22]:

```
    age  workclass     education  educational-num      marital-status  \
0  25.0    Private          11th              7.0       Never-married
1  38.0    Private       HS-grad              9.0  Married-civ-spouse
```

```
2  28.0  Local-gov    Assoc-acdm            12.0  Married-civ-spouse
3  44.0    Private  Some-college            10.0  Married-civ-spouse
4  18.0    Unknown  Some-college            10.0       Never-married

          occupation relationship   race  gender  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male           0.0           0.0
1    Farming-fishing      Husband  White    Male           0.0           0.0
2    Protective-serv      Husband  White    Male           0.0           0.0
3  Machine-op-inspct      Husband  Black    Male        7688.0           0.0
4            Unknown    Own-child  White  Female           0.0           0.0

   hours-per-week native-country income altersgruppe
0            40.0  United-States  <=50K  Middle-aged
1            50.0  United-States  <=50K  Middle-aged
2            40.0  United-States   >50K  Middle-aged
3            40.0  United-States   >50K  Experienced
4            30.0  United-States  <=50K        Young
```

[23]:

```python
import pandas as pd
from sklearn import preprocessing  # WICHTIG: preprocessing importieren

# Kopie der Daten vor der Kodierung erstellen
df1 = df.copy()

# Dictionary zum Speichern der Encoder für jede Spalte
encoders = {}

# Label Encoding für jede kategoriale Spalte
categorical_columns = ['gender', 'workclass', 'education', 'marital-status',
                       'occupation', 'relationship', 'race', 'native-country',␣
 ↪'income']

for col in categorical_columns:
    encoders[col] = preprocessing.LabelEncoder()  # Neuen Encoder für jede␣
 ↪Spalte erstellen
    df[col] = encoders[col].fit_transform(df1[col])  # Werte transformieren und␣
 ↪speichern

# Korrekte Mapping-Werte für jede Spalte anzeigen
for col, encoder in encoders.items():
    print(f"Mapping für {col}:")
    mapping = dict(zip(encoder.classes_, encoder.transform(encoder.classes_))) ␣
 ↪# Mapping-Dictionary erstellen
    for category, code in mapping.items():
        print(f"{code}: {category}")
    print("\n")
```

```
Mapping für gender:
0: Female
1: Male


Mapping für workclass:
0: Federal-gov
1: Local-gov
2: Never-worked
3: Private
4: Self-emp-inc
5: Self-emp-not-inc
6: State-gov
7: Unknown
8: Without-pay


Mapping für education:
0: 10th
1: 11th
2: 12th
3: 5th-6th
4: 7th-8th
5: 9th
6: Assoc-acdm
7: Assoc-voc
8: Bachelors
9: Doctorate
10: HS-grad
11: Masters
12: Prof-school
13: Some-college


Mapping für marital-status:
0: Divorced
1: Married-AF-spouse
2: Married-civ-spouse
3: Married-spouse-absent
4: Never-married
5: Separated
6: Widowed


Mapping für occupation:
0: Adm-clerical
1: Armed-Forces
2: Craft-repair
```

```
3: Exec-managerial
4: Farming-fishing
5: Handlers-cleaners
6: Machine-op-inspct
7: Other-service
8: Priv-house-serv
9: Prof-specialty
10: Protective-serv
11: Sales
12: Tech-support
13: Transport-moving
14: Unknown


Mapping für relationship:
0: Husband
1: Not-in-family
2: Other-relative
3: Own-child
4: Unmarried
5: Wife


Mapping für race:
0: Amer-Indian-Eskimo
1: Asian-Pac-Islander
2: Black
3: Other
4: White


Mapping für native-country:
0: Cambodia
1: Canada
2: China
3: Columbia
4: Cuba
5: Dominican-Republic
6: Ecuador
7: El-Salvador
8: England
9: France
10: Germany
11: Greece
12: Guatemala
13: Haiti
14: Honduras
15: Hong
```

```
16: Hungary
17: India
18: Iran
19: Ireland
20: Italy
21: Jamaica
22: Japan
23: Laos
24: Mexico
25: Nicaragua
26: Outlying-US(Guam-USVI-etc)
27: Peru
28: Philippines
29: Poland
30: Portugal
31: Puerto-Rico
32: Scotland
33: South
34: Taiwan
35: Thailand
36: Trinadad&Tobago
37: United-States
38: Unknown
39: Vietnam
40: Yugoslavia


Mapping für income:
0: <=50K
1: >50K
```

[24]:
```
import pandas as pd
from sklearn import preprocessing  # WICHTIG: preprocessing importieren

# LabelEncoder initialisieren
label_encoder = preprocessing.LabelEncoder()

# Kopie des ursprünglichen DataFrames erstellen (falls nicht bereits geschehen)
df1 = df.copy()

# Label Encoding für jede kategoriale Spalte
categorical_columns = ['gender', 'workclass', 'education', 'marital-status',
                       'occupation', 'relationship', 'race', 'native-country',␣
 ↪'income']
```

```
for col in categorical_columns:
    df1[col] = label_encoder.fit_transform(df1[col])

# Die ersten Zeilen nach Label-Encoding anzeigen
df1.head()
```

[24]:

```
    age  workclass  education  educational-num  marital-status  occupation  \
0  25.0          3          1              7.0               4           6
1  38.0          3         10              9.0               2           4
2  28.0          1          6             12.0               2          10
3  44.0          3         13             10.0               2           6
4  18.0          7         13             10.0               4          14

   relationship  race  gender  capital-gain  capital-loss  hours-per-week  \
0             3     2       1           0.0           0.0            40.0
1             0     4       1           0.0           0.0            50.0
2             0     4       1           0.0           0.0            40.0
3             0     2       1        7688.0           0.0            40.0
4             3     4       0           0.0           0.0            30.0

   native-country  income altersgruppe
0              37       0  Middle-aged
1              37       0  Middle-aged
2              37       1  Middle-aged
3              37       1  Experienced
4              37       0        Young
```

[25]:

```
# writing to a csv file
# df1 is data after processed

df1.to_csv('../data/raw/adult-income-processed.csv', index=False)
```

The label encoded data is written to csv file.

The python for data preprocessing is to be found in through the link "https://github.com/fhswf-study-projects/mlops-data-processor/blob/ea71291690b642aff65f3f85bf09c309c8948099/src/data_preprocessing.py"

The data preprocessing file starts with cleaning and feature creation, done before the pipeline runs. First, unnecessary columns like "fnlwgt" are removed to keep things simple. Missing values in important columns like "workclass", "occupation", and "native_country" are filled with the word "Unknown" so that no rows are lost.

Next, new features are created to improve the data. An "age_group" column is added, grouping people into categories like "Young" and "Middle-aged". The target column "income" is also converted into a binary format to show whether someone earns more or less than $50,000.

Inside the pipeline, the data is split into numerical and categorical columns. Numerical columns include things like "age", "education_num", "capital_gain", "capital_loss", and "hours_per_week". Categorical columns include "workclass", "education", "marital_status", "occupation", "relationship", "race", "sex", "native_country", and the new "age_group".

For numerical data, StandardScaler is used to adjust the numbers so they all have a similar scale — this helps the model learn better and faster.

Categorical data is transformed using OneHotEncoding, which turns each category into its own column. For example, the "education" column becomes multiple columns like "education_Bachelors", "education_Masters", and so on, marked with 1 or 0. If new data comes in with a category the model hasn't seen before, handle_unknown="ignore" ensures it's safely ignored instead of causing an error.

Finally, ColumnTransformer combines everything. It applies scaling to numerical data and encoding to categorical data at the same time. This ensures all the data is in the right format and ready to go into the model without extra work.

[ ]:

```

```