01_Data_Exploration_und_Validation

March 16, 2025

1 MLOps Project work: Datenexploration & Validierung

1.1 1. Introduction

In this notebook, we will focus on the data exploration and the data validation. This is a fundamental first step in the MLOps pipeline, as understanding our data and ensuring its quality influences all subsequent steps.

1.2 2. Objectives

In this notebook, we will show the following steps by using the sample dataset 'adult income':

- a structured approach to data exploration
- methods for data validation with Great Expectations
- methods to create a basic data profile
- the importance of data quality in the MLOps context

1.3 3. Theoretical foundations

1.3.1 3.1 Importance of data exploration in MLOps

Data exploration is crucial for several reasons:

1. **Data understanding**
    - Erkennen von Mustern und Zusammenhängen
    - Identifizierung von Ausreißern
    - Verständnis der Datenverteilungen
2. **Quality assurance**
    - Detecting data problems
    - Checking for data completeness
    - Identifying inconsistencies in the data
3. **Business context**
    - Reviewing business requirements
    - Identifying relevant features
    - Understanding the domain

1.3.2 3.2 Systematic approach

For effective data exploration, we will use the following approach:

1. **Very first analysis**

- Overview of the data structure
  - Checking the data types
  - Identifying missing values
2. **Detailed investigation**
   - Statistical key figures
   - Distribution analyses
   - Correlation analyses
3. **Quality check**
   - Data validation
   - Consistency checks
   - Documentation of anomalies

1.4 4. Practical implementation

1.4.1 4.1 Importing necessary libraries

[2]:
```
# import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import great_expectations as ge
#from great_expectations.dataset import PandasDataset

#  outputs for plots
plt.rcParams['axes.formatter.use_locale'] = True
plt.style.use('seaborn-v0_8-darkgrid')
```

1.4.2 4.2 Datensatz laden

[4]:
```
# adult income Datensatz laden
# https://raw.githubusercontent.com/fhswf-study-projects/mlops-data-processor/
↪refs/heads/Datenexploration/adult.csv
df = pd.read_csv('../data/raw/adult-income.csv')
print(f"Dataset loaded successfully. Form: {df.shape}")
```

```
Dataset loaded successfully. Form: (48842, 15)
```

1.4.3 4.3 First data inspection

An initial data inspection is performed.

[14]:
```
df.info()
df.dtypes

df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
```

```
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   age              48842 non-null  int64 
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64 
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64 
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64 
 11  capital-loss     48842 non-null  int64 
 12  hours-per-week   48842 non-null  int64 
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

[14]:

```
   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          ?  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                  ?    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  
```

1.4.4 4.4 Exploratory data analysis

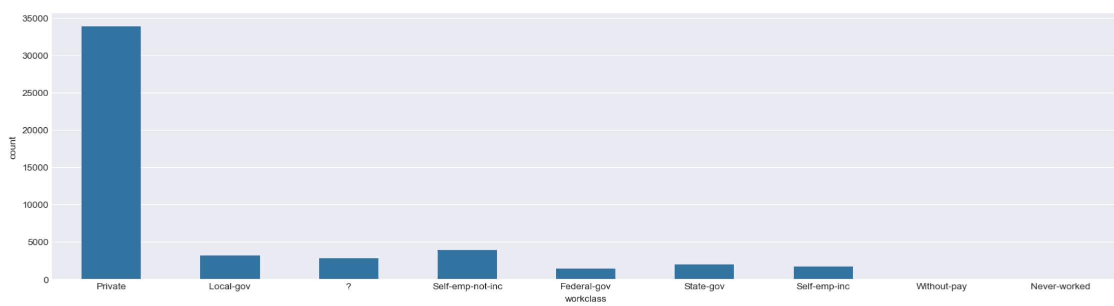
We will perform an exploratory data analysis and focus on categorical as well as numerical variables.

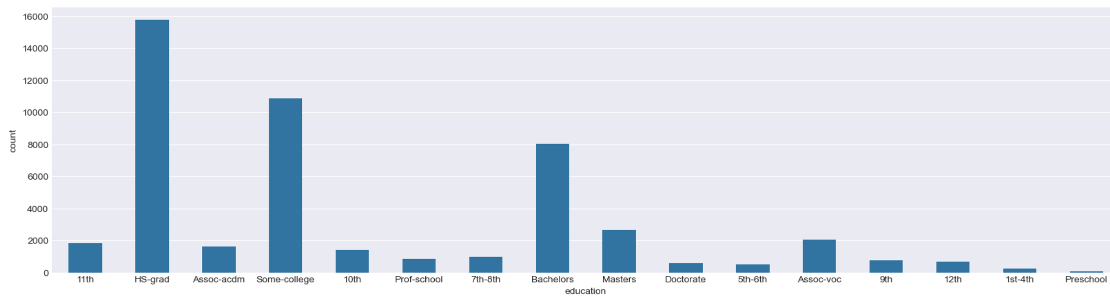
4.4.1 Analyze categorical variables Firstly, we take a look at the categorical variables by creating bar charts for each categorical variable. This section shows the process of exploring the data step by step.

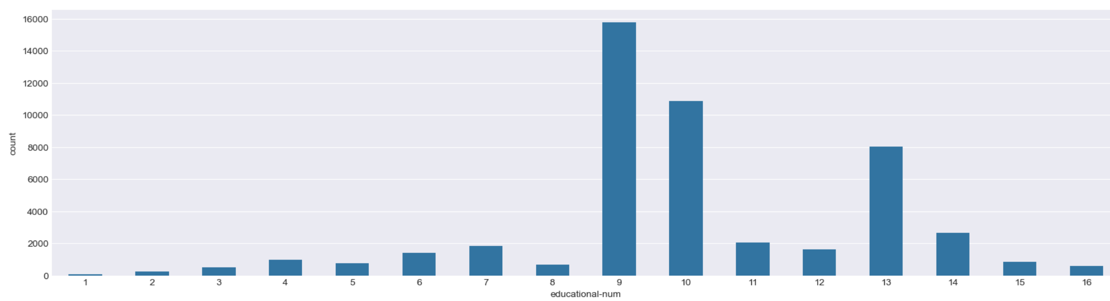
[15]:
```
# identify categorical columns
kategorische_spalten = df.select_dtypes(include=['object']).columns
kategorische_spalten
```

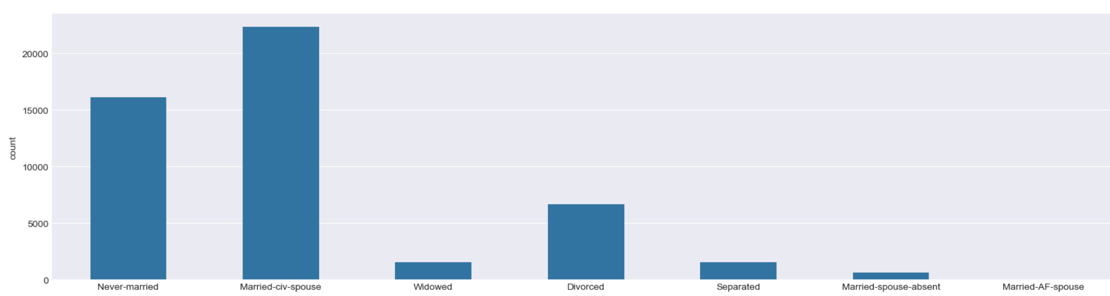
[15]:
```
Index(['workclass', 'education', 'marital-status', 'occupation',
       'relationship', 'race', 'gender', 'native-country', 'income'],
      dtype='object')
```

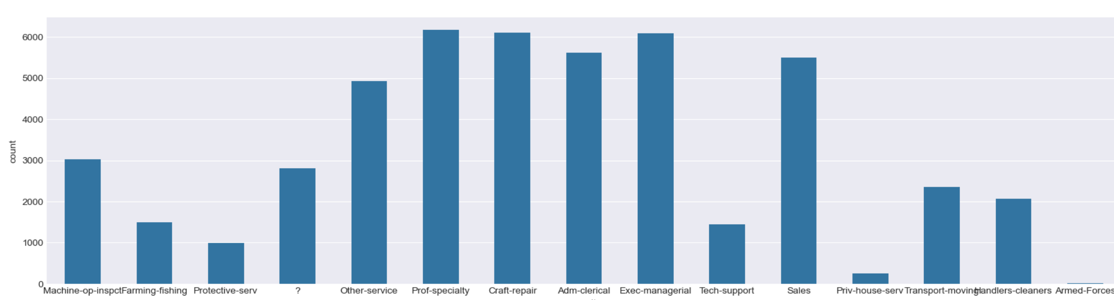
[18]:
```
# Visualize distributions of categorical variables
for col in df.columns:
  if len(df[col].unique()) <20 :
    plt.figure(figsize=(20,5))
    sns.countplot(x=col, data=df,width=0.5)
plt.show()




# too many distinct values in native-countries
# to prevent overlapping of labels, native-countries is plotted separately
country_counts = df['native-country'].value_counts()
plt.figure(figsize=(20, 10))
country_counts.plot(kind='barh')
plt.title('Counts of Native Countries')
plt.xlabel('Number of Individuals')
plt.ylabel('Native Country')

plt.xticks(rotation=45, ha='right')  # Rotate 45 degrees, align right
# or
plt.xticks(rotation=90) # Rotate 90 degrees

plt.tight_layout() # Important to adjust layout to prevent labels from being
 ↪cut off
plt.show()
```

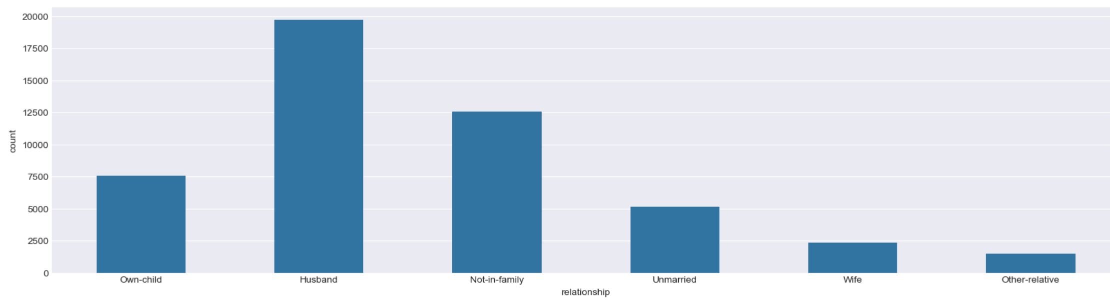


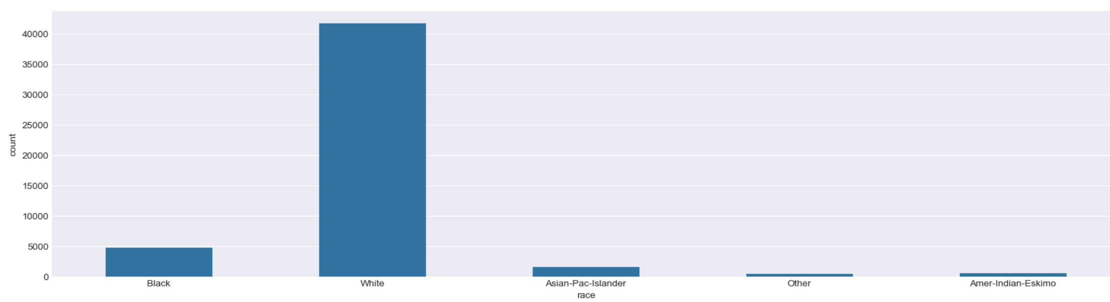

count
education
11th
HS-grad
Assoc-acdm
Some-college
10th
Prof-school
7th-8th
Bachelors
Masters
Doctorate
5th-6th
Assoc-voc
9th
12th
1st-4th
Preschool

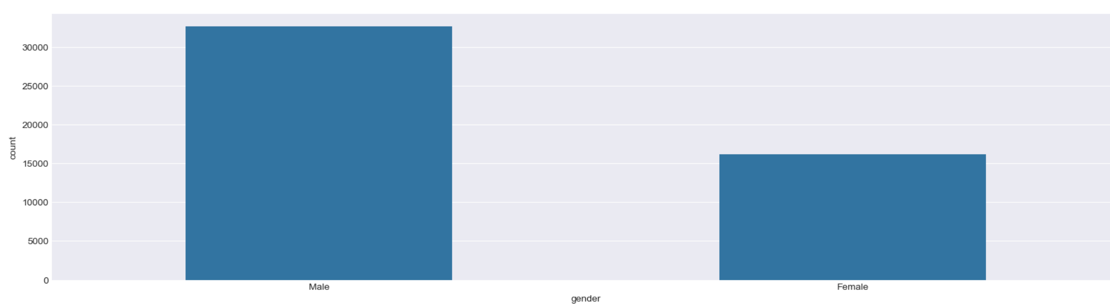

count
educational-num

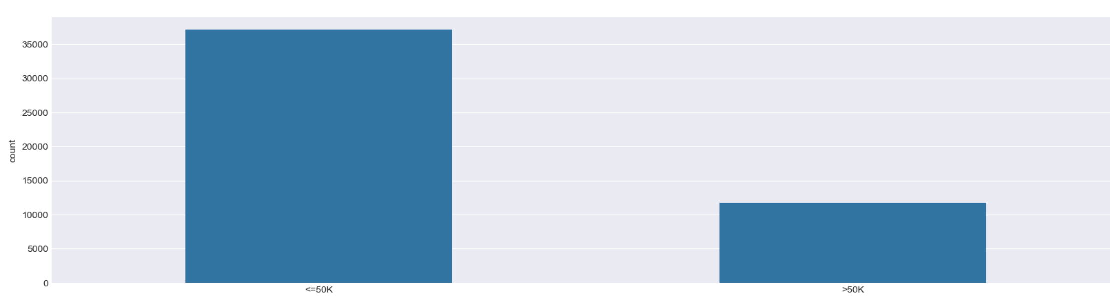

count
marital-status
Never-married
Married-civ-spouse
Widowed
Divorced
Separated
Married-spouse-absent
Married-AF-spouse

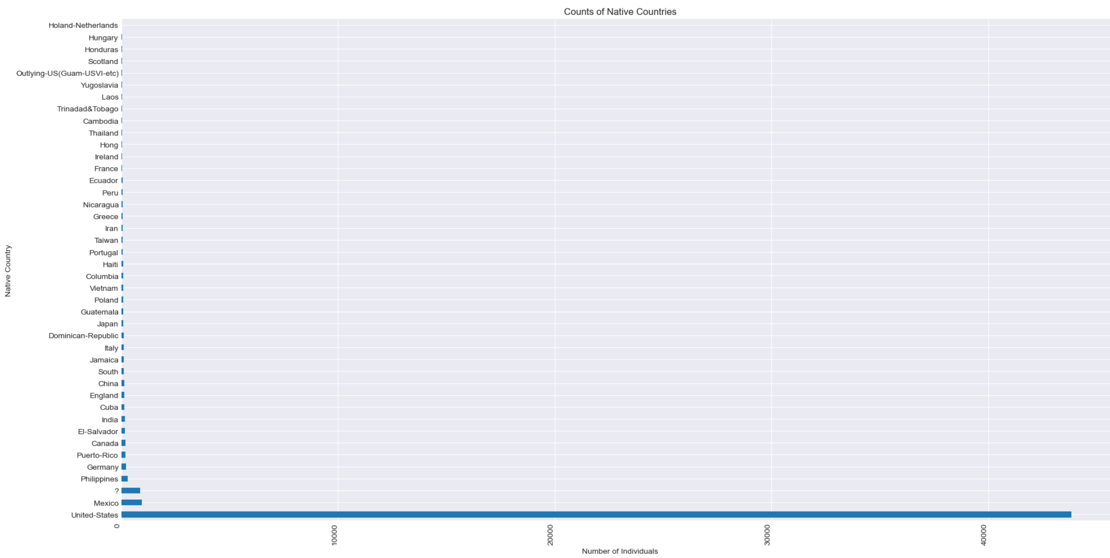

count
occupation
Machine-op-inspct
Farming-fishing
Protective-serv
?
Other-service
Prof-specialty
Craft-repair
Adm-clerical
Exec-managerial
Tech-support
Sales
Priv-house-serv
Transport-moving
Handlers-cleaners
Armed-Forces


20000
17500
15000
12500
10000
7500
5000
2500
0
count
Own-child
Husband
Not-in-family
Unmarried
Wife
Other-relative
relationship


40000
35000
30000
25000
20000
15000
10000
5000
0
count
Black
White
Asian-Pac-Islander
Other
Amer-Indian-Eskimo
race


30000
25000
20000
15000
10000
5000
0
count
Male
Female
gender

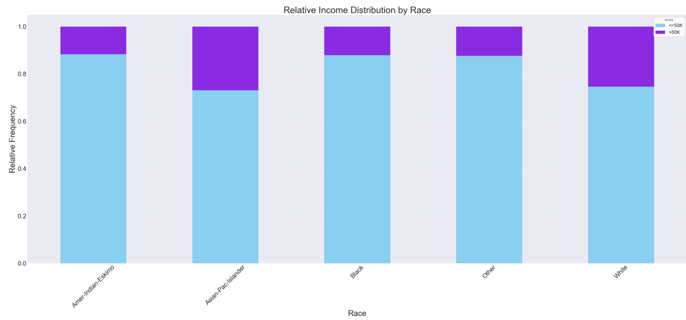

35000
30000
25000
20000
15000
10000
5000
0
count
<=50K
>50K
income

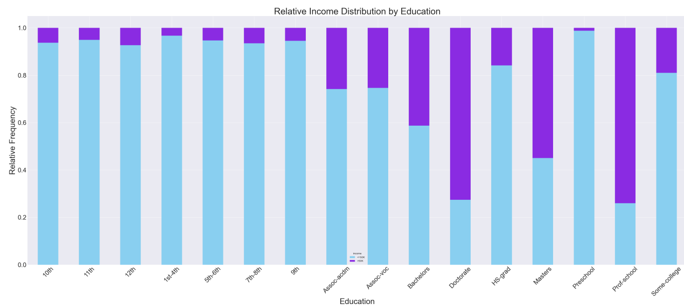


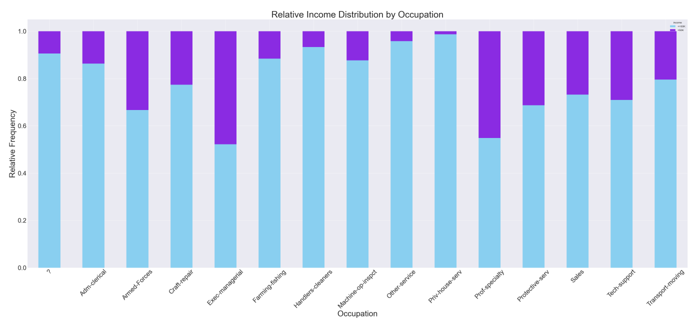
We are going to explore how the income is distributed across the variables.

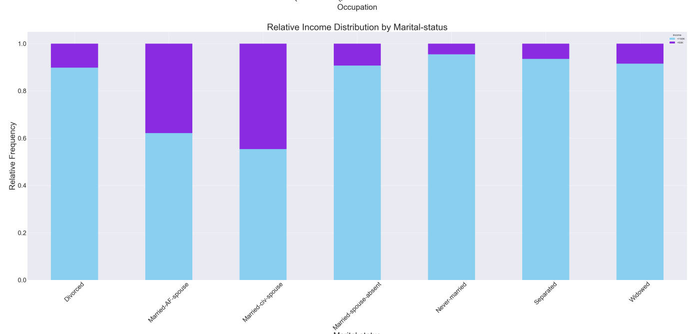
[31]:
```
# Group by race and income to calculate absolute frequencies
income_distribution = df.groupby(["race", "income"]).size().unstack()

# Convert to relative frequencies
income_distribution = income_distribution.div(income_distribution.sum(axis=1),␣
 ↪axis=0)

# Define colors: Light Blue for <=50K, Purple for >50K
colors = ["#89CFF0", "#8A2BE2"]

# Create stacked bar chart for income distribution by race
fig, axes = plt.subplots(6, 1, figsize=(36, 100))  # 6 rows, 1 columns layout
axes = axes.flatten()  # Flatten axes for easy indexing

# Define features to visualize
features = ["race", "education", "occupation", "marital-status", "age",␣
 ↪"workclass"]

for i, feature in enumerate(features):
    feature_distribution = df.groupby([feature, "income"]).size().unstack()
    feature_distribution = feature_distribution.div(feature_distribution.
 ↪sum(axis=1), axis=0)

    # Plot stacked bar chart
    feature_distribution.plot(kind="bar", stacked=True, color=colors,␣
 ↪ax=axes[i])
```

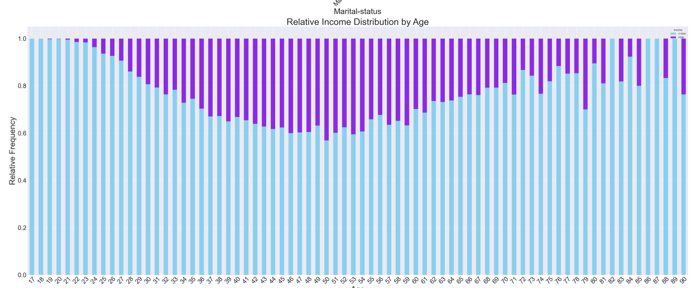
```python
    # Set titles and labels with larger font size
    axes[i].set_title(f"Relative Income Distribution by {feature.
 ↪capitalize()}", fontsize=34)
    axes[i].set_xlabel(feature.capitalize(), fontsize=30)
    axes[i].set_ylabel("Relative Frequency", fontsize=30)
    axes[i].tick_params(axis="x", rotation=45, labelsize=24)  # Rotate and␣
 ↪increase size of x-ticks
    axes[i].tick_params(axis="y", labelsize=24)  # Increase size of y-ticks

# Customize legend with a white background and larger font size
legend = axes[0].legend(title="Income", labels=["<=50K", ">50K"], frameon=True,␣
 ↪fontsize=18)
legend.get_frame().set_facecolor("white")

# Adjust layout for better readability
plt.tight_layout()
plt.show()
```

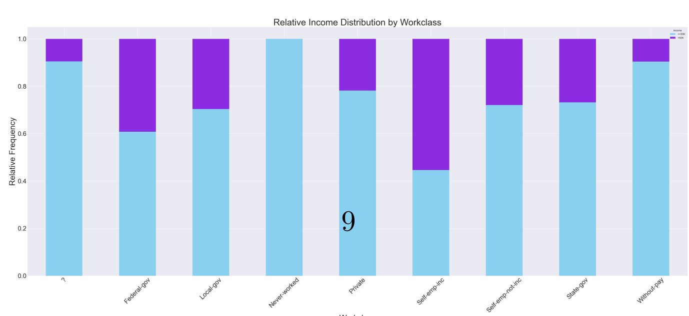

Relative Income Distribution by Race
Relative Frequency
Amer-Indian-Eskimo
Asian-Pac-Islander
Black
Other
White
Race


Relative Income Distribution by Education
Relative Frequency
10th
11th
12th
1st-4th
5th-6th
7th-8th
9th
Assoc-acdm
Assoc-voc
Bachelors
Doctorate
HS-grad
Masters
Preschool
Prof-school
Some-college
Education


Relative Income Distribution by Occupation
Relative Frequency
?
Adm-clerical
Armed-Forces
Craft-repair
Exec-managerial
Farming-fishing
Handlers-cleaners
Machine-op-inspct
Other-service
Priv-house-serv
Prof-specialty
Protective-serv
Sales
Tech-support
Transport-moving
Occupation


Relative Income Distribution by Marital-status
Relative Frequency
Divorced
Married-AF-spouse
Married-civ-spouse
Married-spouse-absent
Never-married
Separated
Widowed
Marital-status


Relative Income Distribution by Age
Relative Frequency
Age

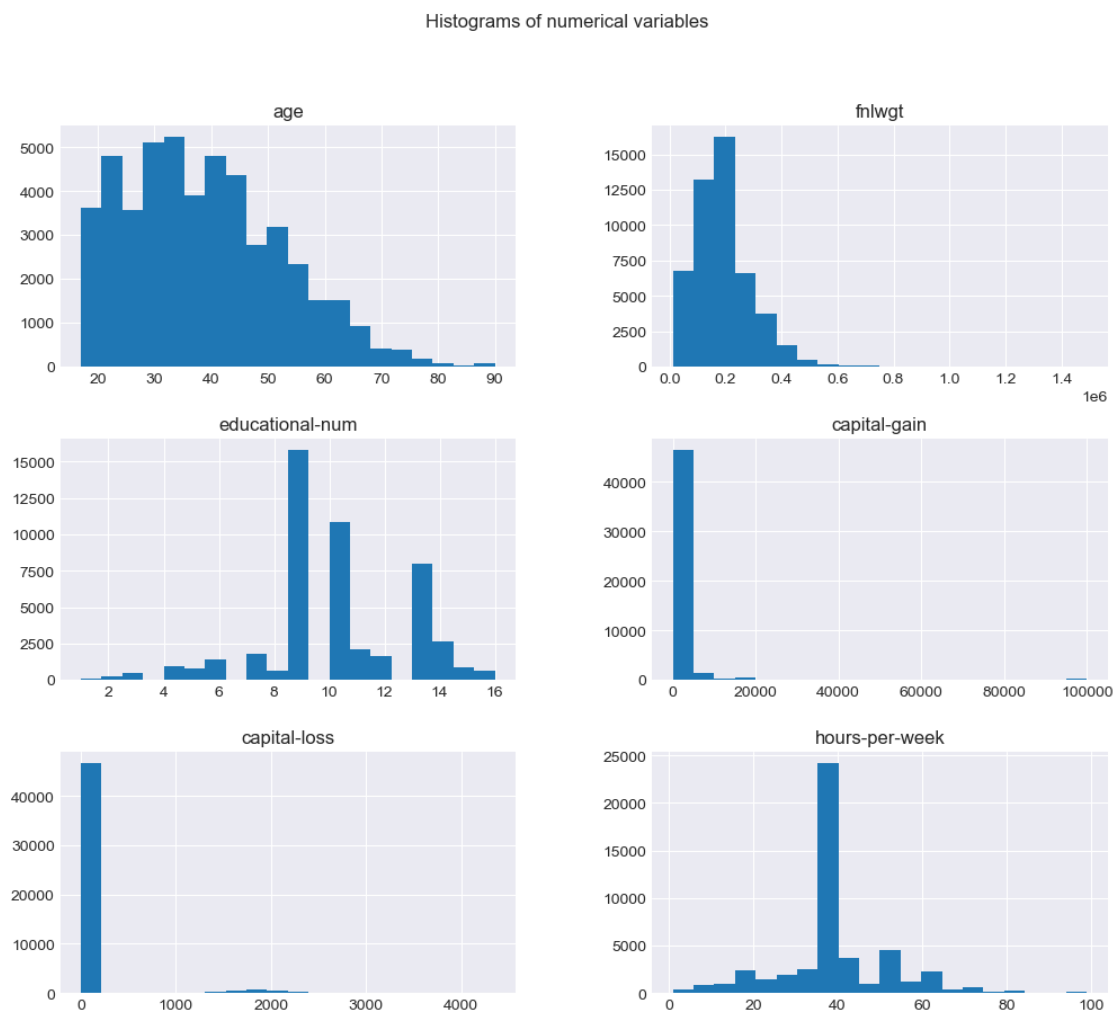

Relative Income Distribution by Workclass
Relative Frequency
?
Federal-gov
Local-gov
Never-worked
Private
Self-emp-inc
Self-emp-not-inc
State-gov
Without-pay
Workclass

4.4.2 Identify numerical variables Here, we will analyze the numerical variables by creating histograms for numerical variables and also a correlation matrix.

[15]:
```
# Identifizierung numerischer Spalten
numerische_spalten = data.select_dtypes(include=['int64', 'float64']).columns
numerische_spalten
```

[15]:
```
Index(['age', 'fnlwgt', 'educational-num', 'capital-gain', 'capital-loss',
       'hours-per-week'],
      dtype='object')
```

[16]:
```
# create histogramm

data.hist(figsize=(12, 10), bins=20)
plt.suptitle("Histograms of numerical variables")
plt.show()
```



The dataset has 48842 entries with each entry represents a person surveyed. There are 15 columns/features and the target feature is income. It defines whether a person's annual year (income exceeds or not) based on census data.

Features types are: - Numerical features (6 features, int64): age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week - Categorical features (9 features, object(str)): workclass, education, marital-status, occupation, relationship, race, sex, native-country, income

There are no missing data or no null data in any of the dataset columns. However, there are unknown data in columns namely 'workclass' 'occupation' 'native-country' indicated by the symbol '?'. While occupation is further classified into workclass. For example, machine-op-inspection, farming-fishing, craft-repair etc are classified into private workclass.

Based on the data set, we discovered that duplicated columns convey the same information such as "education" and "educational-num". Both are in fact same information. Therefore, one column can dropped and the classification of educational levels can be further aggregated. Furthermore, some columns have unnecessary detail grade, for example race. The data column could be summarized into two main groups "white" and "non-white", as the race groups other than whites are already underrepresented in the dataset.

1.4.5 4.5 Data validation with GX (great expectations)

4.5.1 Create GX dataset

[17]:
```
# Pandas DataFrame in Great Expectations Dataset umwandeln
context = ge.get_context()
data_source = context.data_sources.add_pandas(name = "my_pandas_datasource")
data_asset = data_source.add_dataframe_asset(name = "my_dataframe_asset")
batch_definition = data_asset.add_batch_definition_whole_dataframe(name =␣
  ↪"my_batch_definition")
batch = batch_definition.get_batch(batch_parameters={"dataframe": data})
```

[18]:
```
df.shape
```

[18]:
```
(48842, 15)
```

[19]:
```
row_count_definition = ge.expectations.ExpectTableRowCountToEqual(value = 48842)
null_expectation = ge.expectations.
  ↪ExpectColumnValuesToNotBeNull(column="income")
#unique_expectations = ge.expectations.
  ↪ExpectColumnValuesToBeUnique(column="customerID")


validation_results = batch.validate(expect= row_count_definition)
#validation_results = batch.validate(expect= null_expectation)
#validation_results = batch.validate(expect= unique_expectations)
```

```
print(validation_results.success)
```

```
Calculating Metrics:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
True
```

4.5.2 Defining basic expectations Define basic expectations for the data set. This is divided into two parts:

- data type and structure expectations
- value range and distribution expectations

[20]:
```
# data type and structure expectations

columnExist_expectation = ge.expectations.ExpectColumnToExist(column="income")
validation_results = batch.validate(expect= columnExist_expectation)
print(validation_results.success)
```

```
Calculating Metrics:   0%|          | 0/2 [00:00<?, ?it/s]
```

```
True
```

[21]:
```
# data type and structure expectations

NotNull_expectation = ge.expectations.
 ↪ExpectColumnValuesToNotBeNull(column="income")
validation_results = batch.validate(expect= NotNull_expectation)
print(validation_results.success)
```

```
Calculating Metrics:   0%|          | 0/8 [00:00<?, ?it/s]
```

```
True
```

[22]:
```
# data type and structure expectations
# values must be between min/max set

# assume that no one is still working at the age of 80
BetweenAge_expectation = ge.expectations.ExpectColumnValuesToBeBetween(
    column="age", min_value=18, max_value=80
)

# it is impossible to work more than 168 hours per week
BetweenHoursPerWeek_expectation = ge.expectations.ExpectColumnValuesToBeBetween(
    column="hours-per-week", min_value=8, max_value=168
)


validation_resultsBetweenAge = batch.validate(expect= columnExist_expectation)
validation_resultsBetweenHoursPerWeek = batch.validate(expect=␣
 ↪columnExist_expectation)
```

```
# print results

print(validation_resultsBetweenAge.success)
print(validation_resultsBetweenHoursPerWeek.success)
```

```
Calculating Metrics:   0%|          | 0/2 [00:00<?, ?it/s]

Calculating Metrics:   0%|          | 0/2 [00:00<?, ?it/s]

True
True
```

Secondly, the statistical expectations are checked.

[23]:
```
# assuming start working age at 18 and retirement age of 80. Our dataset is␣
↪skewed towards younger employees.
# The median should be around 30-40 years old.

expectation_Median = ge.expectations.ExpectColumnMedianToBeBetween(
    column="age", min_value=30, max_value=40
)

validation_results = batch.validate(expect= expectation_Median)
print(validation_results.success)
```

```
Calculating Metrics:   0%|          | 0/4 [00:00<?, ?it/s]

True
```

[ ]:
```
```