01_tabular_data_exploration

May 29, 2021

1 First look at our dataset

In this notebook, we will look at the necessary steps required before any machine learning takes place. It involves:

- loading the data;
- looking at the variables in the dataset, in particular, differentiate between numerical and categorical variables, which need different preprocessing in most machine learning workflows;
- visualizing the distribution of the variables to gain some insights into the dataset.

1.1 Loading the adult census dataset

We will use data from the 1994 US census that we downloaded from OpenML.

You can look at the OpenML webpage to learn more about this dataset: http://www.openml.org/d/1590

The dataset is available as a CSV (Comma-Separated Values) file and we will use pandas to read it.

Note

Pandas is a Python library used for manipulating 1 and 2 dimensional structured data. If you have never used pandas, we recommend you look at this tutorial.

[1]:
```
import pandas as pd

adult_census = pd.read_csv("../datasets/adult-census.csv")
```

The goal with this data is to predict whether a person earns over 50K a year from heterogeneous data such as age, employment, education, family information, etc.

1.2 The variables (columns) in the dataset

The data are stored in a pandas dataframe. A dataframe is a type of structured data composed of 2 dimensions. This type of data is also referred as tabular data.

Each row represents a sample. In the field of machine learning or descriptive statistics, commonly used equivalent terms are “record”, “instance”, or “observation”.

Each column represents a type of information that has been collected and is called a feature. In the field of machine learning and descriptive statistics, commonly used equivalent terms are “variable”, "attribute", or "covariate".

A quick way to inspect the dataframe is to show the first few lines with the `head` method:

[2]:
```
adult_census.head()
```

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

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

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

The column named **class** is our target variable (i.e., the variable which we want to predict). The two possible classes are `<=50K` (low-revenue) and `>50K` (high-revenue). The resulting prediction problem is therefore a binary classification problem, while we will use the other columns as input variables for our model.

[3]:
```
target_column = 'class'
adult_census[target_column].value_counts()
```

[3]:
```
 <=50K    37155
 >50K     11687
Name: class, dtype: int64
```

Note

Classes are slightly imbalanced, meaning there are more samples of one or more classes compared to others. Class imbalance happens often in practice and may need special techniques when building a predictive model.

For example in a medical setting, if we are trying to predict whether subjects will develop a rare disease, there will be a lot more healthy subjects than ill subjects in the dataset.

The dataset contains both numerical and categorical data. Numerical values take continuous values, for example `age`. Categorical values can have a finite number of values, for example native-country.

[4]:
```python
numerical_columns = [
    'age', 'education-num', 'capital-gain', 'capital-loss',
    'hours-per-week']
categorical_columns = [
    'workclass', 'education', 'marital-status', 'occupation',
    'relationship', 'race', 'sex', 'native-country']
all_columns = numerical_columns + categorical_columns + [
    target_column]

adult_census = adult_census[all_columns]
```

We can check the number of samples and the number of columns available in the dataset:

[5]:
```python
print(f"The dataset contains {adult_census.shape[0]} samples and "
      f"{adult_census.shape[1]} columns")
```

```
The dataset contains 48842 samples and 14 columns
```

We can compute the number of features by counting the number of columns and subtract 1, since one of the columns is the target.

[6]:
```python
print(f"The dataset contains {adult_census.shape[1] - 1} features.")
```

```
The dataset contains 13 features.
```

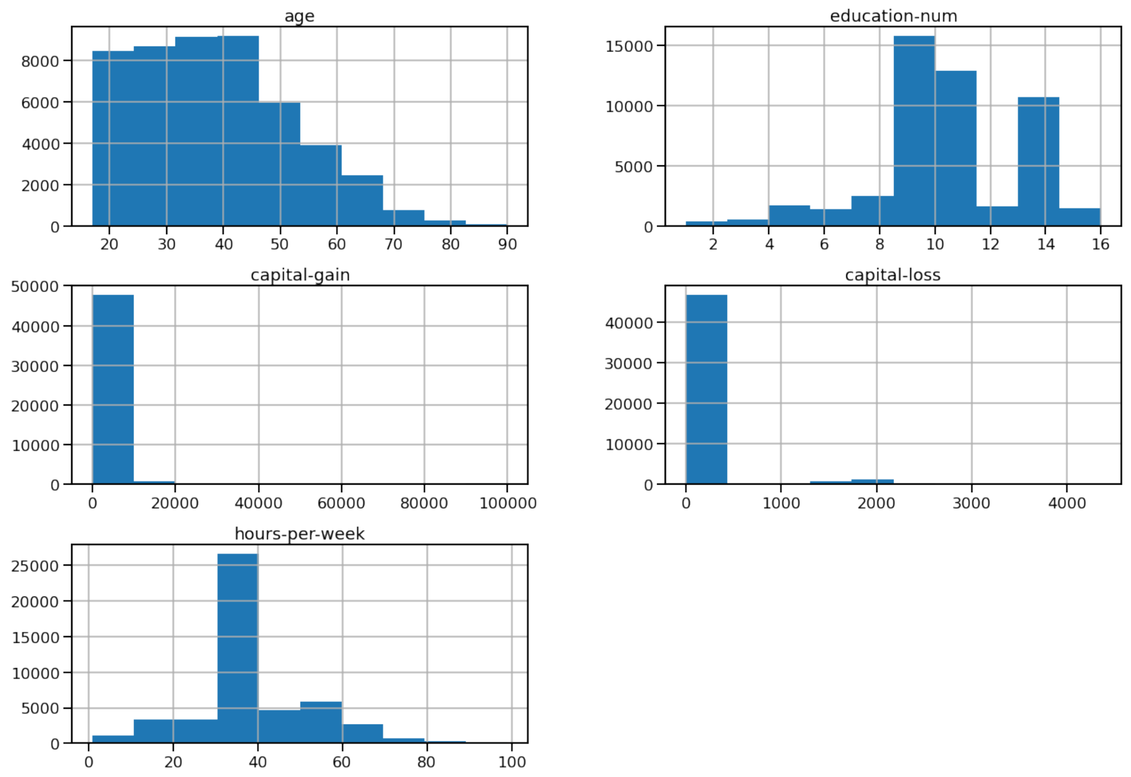
1.3 Visual inspection of the data

Before building a predictive model, it is a good idea to look at the data:

- maybe the task you are trying to achieve can be solved without machine learning;
- you need to check that the information you need for your task is actually present in the dataset;
- inspecting the data is a good way to find peculiarities. These can arise during data collection (for example, malfunctioning sensor or missing values), or from the way the data is processed afterwards (for example capped values).

Let's look at the distribution of individual features, to get some insights about the data. We can start by plotting histograms, note that this only works for features containing numerical values:

[7]:
```python
_ = adult_census.hist(figsize=(20, 14))
```



Tip

In the previous cell, we used the following pattern: _ = func(). We do this to avoid showing the output of func() which in this case is not that useful. We actually assign the output of func() into the variable _ (called underscore). By convention, in Python the underscore variable is used as a "garbage" variable to store results that we are not interested in.

We can already make a few comments about some of the variables:

- `age`: there are not that many points for `age > 70`. The dataset description does indicate that retired people have been filtered out (`hours-per-week > 0`);
- `education-num`: peak at 10 and 13, hard to tell what it corresponds to without looking much further. We'll do that later in this notebook;
- `hours-per-week` peaks at 40, this was very likely the standard number of working hours at the time of the data collection;
- most values of `capital-gain` and `capital-loss` are close to zero.

For categorical variables, we can look at the distribution of values:

[8]:
```
adult_census['sex'].value_counts()
```

[8]:
```
 Male      32650
 Female    16192
Name: sex, dtype: int64
```

```
[9]: adult_census['education'].value_counts()
```

```
[9]:  HS-grad         15784
      Some-college    10878
      Bachelors        8025
      Masters          2657
      Assoc-voc        2061
      11th             1812
      Assoc-acdm       1601
      10th             1389
      7th-8th           955
      Prof-school       834
      9th               756
      12th              657
      Doctorate         594
      5th-6th           509
      1st-4th           247
      Preschool          83
     Name: education, dtype: int64
```

As noted above, `education-num` distribution has two clear peaks around 10 and 13. It would be reasonable to expect that `education-num` is the number of years of education.

Let's look at the relationship between `education` and `education-num`.

```
[10]: pd.crosstab(index=adult_census['education'],
                  columns=adult_census['education-num'])
```

```
[10]: education-num  1    2    3    4    5     6     7    8      9     10    11  \
      education
       10th          0    0    0    0    0  1389     0    0      0      0     0
       11th          0    0    0    0    0     0  1812    0      0      0     0
       12th          0    0    0    0    0     0     0  657      0      0     0
       1st-4th       0  247    0    0    0     0     0    0      0      0     0
       5th-6th       0    0  509    0    0     0     0    0      0      0     0
       7th-8th       0    0    0  955    0     0     0    0      0      0     0
       9th           0    0    0    0  756     0     0    0      0      0     0
       Assoc-acdm    0    0    0    0    0     0     0    0      0      0     0
       Assoc-voc     0    0    0    0    0     0     0    0      0      0  2061
       Bachelors     0    0    0    0    0     0     0    0      0      0     0
       Doctorate     0    0    0    0    0     0     0    0      0      0     0
       HS-grad       0    0    0    0    0     0     0    0  15784      0     0
       Masters       0    0    0    0    0     0     0    0      0      0     0
       Preschool    83    0    0    0    0     0     0    0      0      0     0
       Prof-school   0    0    0    0    0     0     0    0      0      0     0
       Some-college  0    0    0    0    0     0     0    0      0  10878     0

      education-num    12    13    14   15   16
```

```
education
 10th             0     0     0    0    0
 11th             0     0     0    0    0
 12th             0     0     0    0    0
 1st-4th          0     0     0    0    0
 5th-6th          0     0     0    0    0
 7th-8th          0     0     0    0    0
 9th              0     0     0    0    0
 Assoc-acdm    1601     0     0    0    0
 Assoc-voc        0     0     0    0    0
 Bachelors        0  8025     0    0    0
 Doctorate        0     0     0    0  594
 HS-grad          0     0     0    0    0
 Masters          0     0  2657    0    0
 Preschool        0     0     0    0    0
 Prof-school      0     0     0  834    0
 Some-college     0     0     0    0    0
```

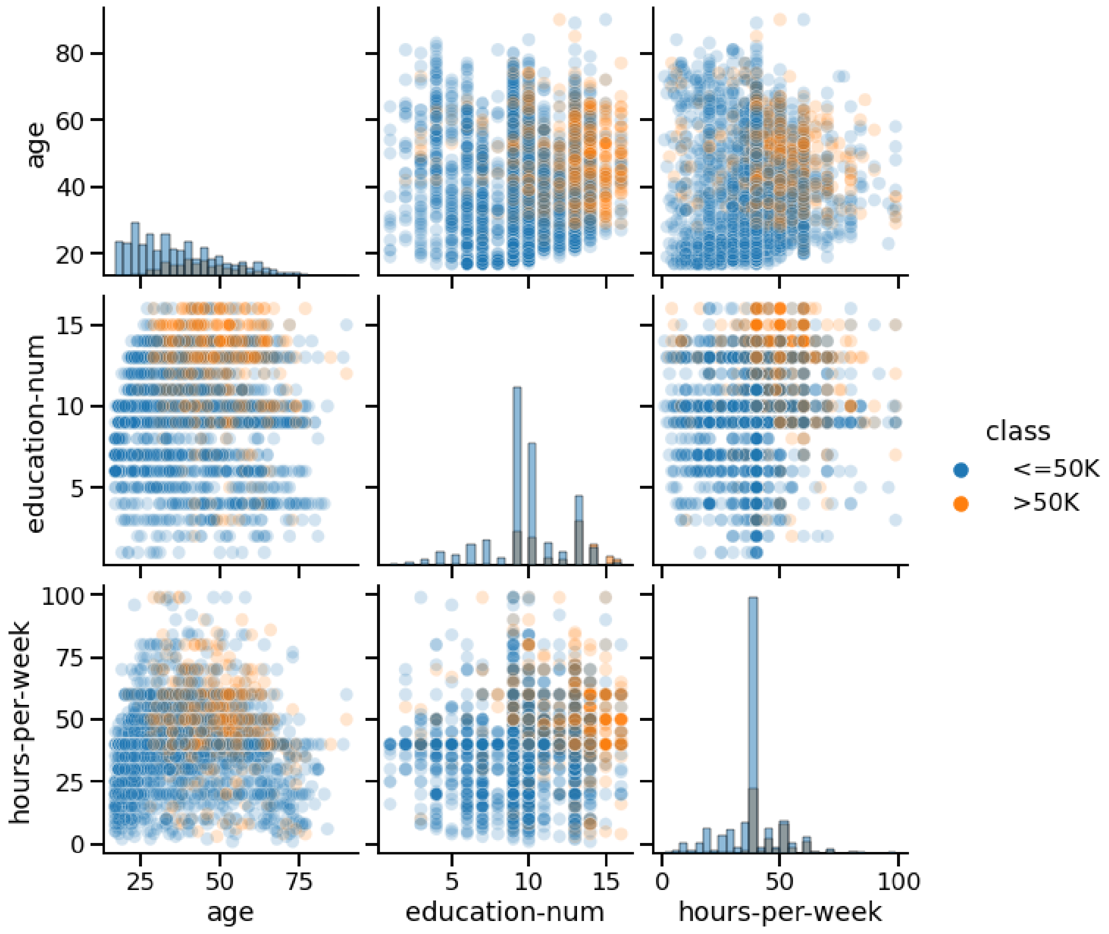
This shows that `education` and `education-num` give you the same information. For example, `education-num=2` is equivalent to `education='1st-4th'`. In practice that means we can remove `education-num` without losing information. Note that having redundant (or highly correlated) columns can be a problem for machine learning algorithms.

Note

In the upcoming notebooks, we will only keep the education variable, excluding the education-num variable since the latter is redundant with the former.

Another way to inspect the data is to do a `pairplot` and show how each variable differs according to our target, i.e. `class`. Plots along the diagonal show the distribution of individual variables for each `class`. The plots on the off-diagonal can reveal interesting interactions between variables.

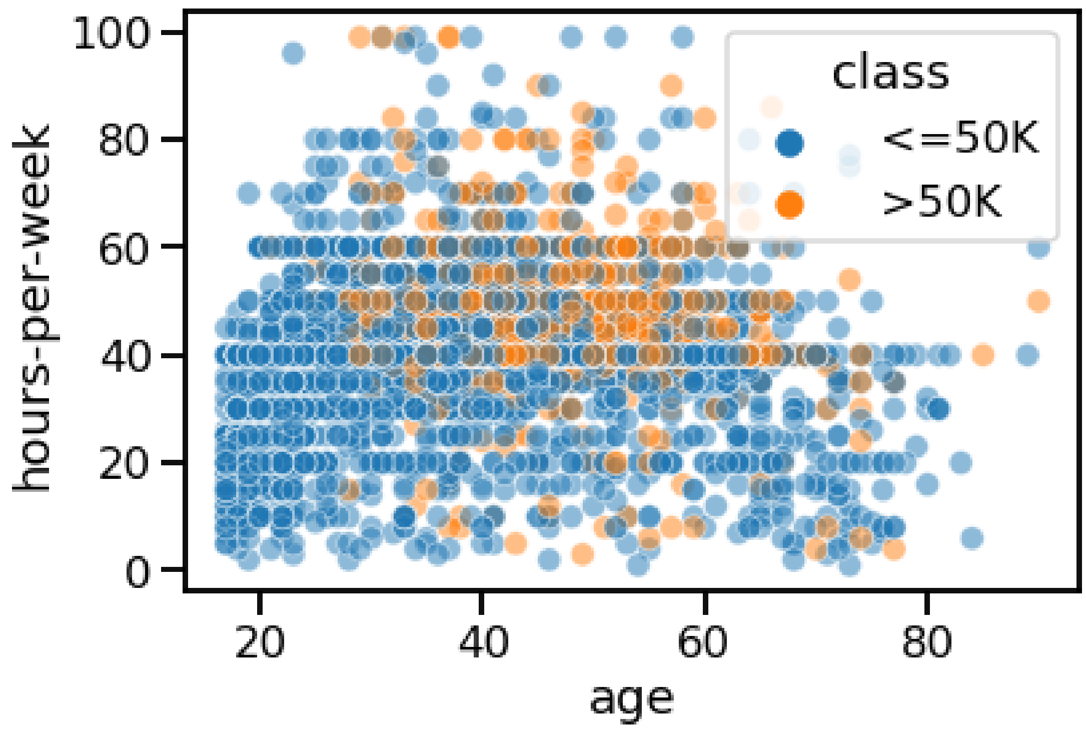
[11]:
```python
import seaborn as sns

# We will plot a subset of the data to keep the plot readable and make the
# plotting faster
n_samples_to_plot = 5000
columns = ['age', 'education-num', 'hours-per-week']
_ = sns.pairplot(data=adult_census[:n_samples_to_plot], vars=columns,
                 hue=target_column, plot_kws={'alpha': 0.2},
                 height=3, diag_kind='hist', diag_kws={'bins': 30})
```



1.4 Creating decision rules by hand

By looking at the previous plots, we could create some hand-written rules that predicts whether someone has a high- or low-income. For instance, we could focus on the combination of the `hours-per-week` and `age` features.

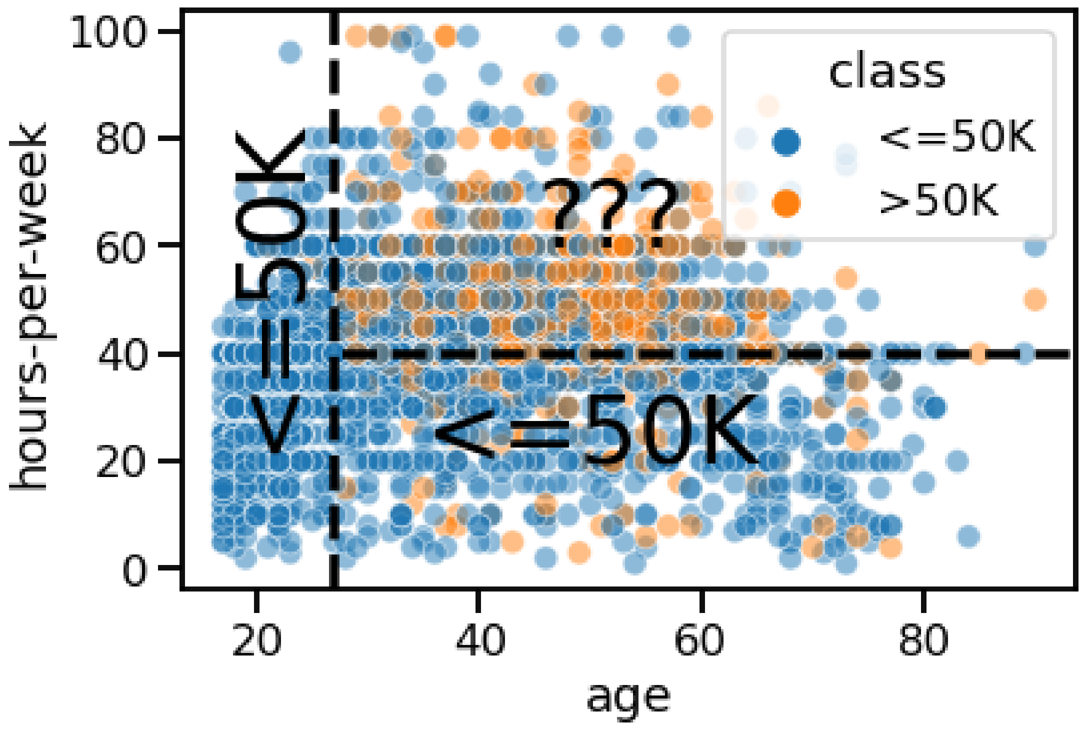
```
[12]: _ = sns.scatterplot(
          x="age", y="hours-per-week", data=adult_census[:n_samples_to_plot],
          hue="class", alpha=0.5,
      )
```



The data points (circles) show the distribution of `hours-per-week` and `age` in the dataset. Blue points mean low-income and orange points mean high-income. This part of the plot is the same as the bottom-left plot in the pairplot above.

In this plot, we can try to find regions that mainly contains a single class such that we can easily decide what class one should predict. We could come up with hand-written rules as shown in this plot:

[13]:
```python
import matplotlib.pyplot as plt

ax = sns.scatterplot(
    x="age", y="hours-per-week", data=adult_census[:n_samples_to_plot],
    hue="class", alpha=0.5,
)

age_limit = 27
plt.axvline(x=age_limit, ymin=0, ymax=1, color="black", linestyle="--")

hours_per_week_limit = 40
plt.axhline(
    y=hours_per_week_limit, xmin=0.18, xmax=1, color="black", linestyle="--"
)

plt.annotate("<=50K", (17, 25), rotation=90, fontsize=35)
plt.annotate("<=50K", (35, 20), fontsize=35)
_ = plt.annotate("???", (45, 60), fontsize=35)
```



- In the region `age < 27` (left region) the prediction is low-income. Indeed, there are many blue points and we cannot see any orange points.
- In the region `age > 27 AND hours-per-week < 40` (bottom-right region), the prediction is low-income. Indeed, there are many blue points and only a few orange points.
- In the region `age > 27 AND hours-per-week > 40` (top-right region), we see a mix of blue points and orange points. It seems complicated to chose which class we should predict in this region.

It is interesting to note that some machine learning models will work similarly to what we did: they are known as decision tree models. The two thresholds that we chose (27 years and 40 hours) are somewhat arbitrary, i.e. we chose them by only looking at the pairplot. In contrast, a decision tree will chose the "best" splits based on data without human intervention or inspection. Decision trees will be covered in more details in a future module.

Note that machine learning is really interesting when creating rules by hand is not straightforward, for example because we are in high dimension (many features) or because there are no simple and obvious rules that separate the two classes as in the top-right region of the previous plot.

To sum up, the important thing to remember is that in a machine-learning setting, a model automatically creates the "rules" from the data in order to make predictions on new unseen data.

In this notebook we have:

- loaded the data from a CSV file using `pandas`;
- looked at the different kind of variables to differentiate between categorical and numerical variables;
- inspected the data with `pandas` and `seaborn`. Data inspection can allow you to decide whether using machine learning is appropriate for your data and to highlight potential peculiarities in your data.

Ideas which will be discussed more in details later:

- if your target variable is imbalanced (e.g., you have more samples from one target category than another), you may need special techniques for training and evaluating your machine learning model;
- having redundant (or highly correlated) columns can be a problem for some machine learning algorithms;
- contrary to decision tree, linear models can only capture linear interaction, so be aware of non-linear relationships in your data.

[ ]:
```

```