Practice Exercise: Cleaning data & Transforming columns/features

Context:

- The data is based on real anonymized Czech bank transactions and account info.
- We'll be focusing on practicing the data cleaning, columns transformations, and other techniques that we've learned in the course.
- But here is the original task description of the dataset publishers:

The bank wants to improve their services. For instance, the bank managers have only vague idea, who is a good client (whom to offer some additional services) and who is a bad client (whom to watch carefully to minimize the bank losses). Fortunately, the bank stores data about their clients, the accounts (transactions within several months), the loans already granted, the credit cards issued. The bank managers hope to improve their understanding of customers and seek specific actions to improve services.

- We've made minor changes on the data to fit this exercise, such as changing the column names. Check out the original source if you are interested in using this data for other purposes (https://data.world/lpetrocelli/czech-financial-dataset-real-anonymized-transactions)

Dataset Description:

We'll work on three datasets (in three separate csv files):

- **account**: each record describes static characteristics of an account
- **transaction**: each record describes one transaction on an account
- **district**: each record describes demographic characteristics of a district

In reality, the organizations like banks often have data stored in multiple datasets. Assume we want to study the transactional level data, we'll need to combine these three datasets together to have transactions data with account and district data.

Objective:

- Examine/clean the individual dataset
- Combine them into a single dataset, which is subject to more cleaning
- Create new columns based on existing columns

By the end, the new dataset is ready for more analysis.

0:40 / 3:31

1. Import the libraries

In [ ]:

2. Import the data from three csv files as DataFrames `account`, `district`, `trans`

Hint:

- the `read_csv` function can automatically infer and load zip file, read its documentation of parameter `compression` if you are interested in details
- you may ignore the warning when reading the `trans.csv.zip` file. It is optional to follow the warning instructions to remove it.

In [ ]:

3. Look at the info summary, head of each DataFrame

In [ ]:

4. Check for the unique values and their counts in each column for the three DataFrames

In [ ]:

5. Check for duplicates in the three DataFrames

In [ ]:

6. Convert column `account_open_date` in `account` and column `date` in `trans` into datetime dtypes

In [ ]:

7. Convert the columns `region` and `district_name` in `district` to all uppercase

2:24 / 3:31

8. Check for missing data by columns in `account` using the `isna` method

In [ ]:

9. Check for missing data by columns in `district` using the `isna` method

In [ ]:

`district` has numeric features that could have relationships with each other. Let's use iterative imputation on them.

Use `IterativeImputer` in `sklearn` to impute based on columns `population`, `average_salary`, `unemployment_rate`, `num_committed_crimes`

Import libraries

In [ ]:

Build a list of columns that will be used for imputation, which are `population`, `average_salary`, `unemployment_rate`, `num_committed_crimes`

These are the columns that might be related to each other

In [ ]:

Create `IterativeImputer` object and set its `min_value` and `max_value` parameters to be the minumum and maximum of corresponding columns

In [ ]:

Apply the imputer to fit and transform the columns to an imputed NumPy array

In [ ]:

Assign the imputed array back to the original DataFrame's columns

2:36 / 3:31

Double check that the columns are imputed

In [ ]:

10. Check for missing data by columns in `trans` using the `isna` method

In [ ]:

Divide the columns into numeric columns and categorical columns, then use the `fillna` method to fill numeric columns with -999, fill categorical columns with 'UNKNOWN'

In [ ]:

11. Check for outliers in `district` using the `describe` method, then look at the histograms of the suspicious columns

In [ ]:

Explore the outliers in the dataset

In [ ]:

12. Check for outliers in `trans` using the `describe` method, then look at the histograms of the suspicious columns

In [ ]:

Explore the outliers in the dataset

In [ ]:

The DataFrame `account` doesn't have any columns that could have outliers, so we are not exploring it.

3:01 / 3:31

The DataFrame `account` doesn't have any columns that could have outliers, so we are not exploring it.

13. Merge (left join) `account` and `district` into a new DataFrame called `account_district` using their common columns

In [ ]:

14. Check the information summary of `account_district`, any missing data?

In [ ]:

Look at the rows with missing data in `account_district`

In [ ]:

Use `SimpleImputer` from `sklearn` to impute the missing data in columns `population`, `average_salary`, `unemployment_rate`, `num_committed_crimes` with their means

In [ ]:

Use `fillna` method to impute the missing data in columns `district_name` and `region` with 'UNKNOWN'

In [ ]:

15. Merge (left join) `trans` and `account_district` into a new DataFrame called `all_data` using their common columns

In [ ]:

Check the information summary of `all_data`

In [ ]:

3:02 / 3:31

Check the information summary of `all_data`

In [ ]:

16. Create a new column `account_open_year` and assign it as the year from column `account_open_date`

In [ ]:

17. Calculate the difference between columns `date` (transaction date) and `account_open_date`

In [ ]:

18. Create a new column `account_age_days` and assign it as the difference in days between columns `date` (transaction date) and `account_open_date`

In [ ]:

19. Create a new column `amount_category` by cutting the column `amount` into 3 equal-sized bins, and label the bins as 'low_amount', 'medium_amount', 'high_amount'

In [ ]:

Verify the categories and their counts in `amount_category`

In [ ]:

20. Create a new column `account_age_days_category` by cutting the column `account_age_days` into 5 equal-width bins

In [ ]:

3:04 / 3:31

20. Create a new column `account_age_days_category` by cutting the column `account_age_days` into 5 equal-width bins

In [ ]:

Verify the categories and their counts in `account_age_days_category`

In [ ]:

Print out the first 20 rows of `all_data` to look at the newly added columns