# Assignment 1, Data Mining

Put all deliverables into github repository in your profile. Share link to google form 24 hours before defense. Defend by explaining deliverables and answering questions. Deliverables: .ipynb Google form: [https://docs.google.com/forms/d/e/1FAIpQLSe0GyNdOYlvM1tX\_I\_CtlPod5jBf-ACLGdHYZq1gV ZbUeBzIg/viewform?usp=sf\_link](https://docs.google.com/forms/d/e/1FAIpQLSe0GyNdOYlvM1tX_I_CtlPod5jBf-ACLGdHYZq1gVZbUeBzIg/viewform?usp=sf_link)

# Exercise 1: Loading Data with Pandas

1. **Objective**: Learn how to load and inspect datasets using Pandas.
2. **Steps**:
   1. Import the Pandas library and load a CSV file into a DataFrame.

○ Use the head(), tail(), and info() functions to inspect the dataset.

○ Check for missing values and data types of each column using isnull() and dtypes.

1. **Questions**:
   1. How do you load a CSV file into a Pandas DataFrame?

To load a CSV file, need to use the pd.read\_csv('file\_name.csv') function.

○ What information does the info() function provide about the dataset?

The info() function displays general information about the data: the number of rows and columns, the column names, the data type in each column, the number of non-empty values, and the amount of memory occupied by the DataFrame.

○ How can you identify missing values in the dataset?

Missing values ​​can be found using the isnull() function. It returns a DataFrame with Boolean values: True if the value is missing and False if it is present.  
df.isnull().sum()

# Exercise 2: Handling Missing Data

1. **Objective**: Practice techniques for handling missing data in a dataset.
2. **Steps**:
   1. Identify missing values in the dataset using isnull().sum().

○ Use different strategies to handle missing data:

■ Remove rows with missing values using dropna().

■ Fill missing values with the mean, median, or a specific value using fillna().

■ Use forward or backward filling (ffill() or bfill()) to fill missing data.

○ Compare the results of each method.

1. **Questions**:
   1. What strategy did you use to handle missing values, and why?

I used the strategy of filling missing values ​​with a specific value (in this case, the date '2024-01-01'). This strategy was chosen to preserve all rows in the dataset and ensure that the gaps are filled with values ​​that make sense in the context of the data. Filling the gaps with a specific value helps preserve the structure of the dataset and avoid information loss, especially if the missing values ​​have a significant number of rows.

○ How did filling missing values affect the dataset?

Filling missing values ​​changes the original data by replacing the missing values ​​with a specified value. This helps avoid problems during further data processing or analysis, since all rows now have values ​​in the correct column. However, it is important to note that filling the missing values ​​with a specific value can distort the results of the analysis, especially if the chosen value does not make sense in the context of the data.

○ When might it be more appropriate to drop rows with missing values instead of filling them?

Removing rows with missing values ​​may be more appropriate in the following cases:

Large number of missing values: If missing values ​​make up a significant portion of the data, filling them in may distort the results of the analysis. In this case, removing rows may be preferable.

Missing in critical columns: If the missing values ​​are in columns that are key to the analysis or modeling, removing them can prevent errors or incorrect conclusions.

High data quality: If the data is important to the accuracy of the analysis and there are other rows without missing values, removing rows with missing values ​​may be justified to maintain high data quality.

# Exercise 3: Data Transformation

1. **Objective**: Transform data to prepare it for analysis.
2. **Steps**:
   1. Normalize numerical features using Min-Max scaling or Z-score standardization with sklearn.preprocessing.

○ Encode categorical variables using one-hot encoding with pd.get\_dummies() or sklearn.preprocessing.OneHotEncoder.

○ Use pd.cut() to bin continuous variables into discrete intervals.

1. **Questions**:
   1. What is the difference between normalization and standardization?

Normalization (Min-Max scaling) scales the data to the range [0, 1], preserving its distribution. This is useful when the data has different scales and you want to bring them to a common range.

Standardization (Z-score) transforms the data so that it has a mean of 0 and a standard deviation of 1. This is useful for normally distributed data or when it is important to account for deviations from the mean.

○ How does one-hot encoding transform categorical variables?

One-hot encoding transforms categorical variables into a set of binary columns (0 or 1), where each column corresponds to one category. This allows categorical variables to be used in machine learning models that require numeric inputs.

○ Why might you want to bin continuous variables into categories?

Grouping continuous variables into categories can be useful for simplifying analysis, identifying patterns, and improving the interpretability of the data. It can also help with models that work better with categorical variables, or in cases where continuous variables have complex distributions and need to be broken down into more understandable groups.

# Exercise 4: Feature Engineering

1. **Objective**: Create new features to improve the predictive power of a dataset.
2. **Steps**:
   1. Create new features by combining or transforming existing features (e.g., adding interaction terms or polynomial features).

○ Extract date-based features (e.g., year, month, day) from datetime columns using pd.to\_datetime() and dt accessor.

○ Use domain knowledge to engineer features that might be useful for your specific problem.

1. **Questions**:
   1. What new features did you create, and why?

In this example, polynomial features were created based on two original features, Feature1 and Feature2. Here are the new features:

Feature1^2: The square of the first feature.

Feature2^2: The square of the second feature.

Feature1 \* Feature2: The product of the two original features. These features were added so that the model could better capture nonlinear dependencies between features. In machine learning problems, such dependencies can be important, and the model can improve its predictions by taking into account feature interactions or their degrees.

○ How did the new features improve the dataset?

Adding new polynomial features can improve a dataset in the following ways:

Accounting for complex dependencies: New features allow the model to capture more complex relationships between variables. For example, if there is a quadratic or multiplicative relationship between two features, polynomial features can help account for this.

Expanding the feature space: This can be useful for algorithms like linear regression or decision trees, which cannot always effectively model complex dependencies based on the original features.

Increasing predictive power: In some cases, polynomial features can improve prediction accuracy because the model can better understand the structure of the data.

However, it is worth remembering that too many features can lead to overfitting the model, especially if the data is small.

○ How can date-based features be useful in a dataset?

Date-based features can add valuable information depending on the task. For example:

Year, month, day of week, hour: These features can reveal seasonal or time-based dependencies. For example, product sales may vary by season or day of the week.

Time since event: This can be useful for analyzing long-term trends (for example, how long has it been since a user registered).

Day of week: A feature indicating what day of the week it is (for example, Monday or Friday) can be useful for demand, traffic, or user behavior forecasting.

Quarter or half-year: In financial data, for example, such features can be useful for analyzing company reports.

Date-based features allow you to take time into account, which is especially important in time series, sales forecasting, user behavior, and other tasks where time patterns can play a key role.

# Exercise 5: Data Cleaning

1. **Objective**: Clean data to ensure it's ready for analysis.
2. **Steps**:
   1. Remove duplicate rows using drop\_duplicates().

○ Detect and remove outliers using the Z-score method or the IQR method.

○ Correct inconsistencies in categorical data (e.g., standardizing text formats or merging similar categories).

1. **Questions**:
   1. How did you identify and handle duplicate rows in the dataset?

To check for duplicates, the df.duplicated() method is used, which returns a series of Boolean values. If a row is repeated, it is marked as True, otherwise - as False. To remove duplicate rows, the drop\_duplicates() method is used. It removes all duplicate rows, leaving only unique rows.

○ What method did you use to detect and remove outliers, and why?

To detect outliers, the interquartile range (IQR) method was used. It is based on the data distribution and calculation of the boundaries beyond which outliers fall.

IQR (Interquartile Range) is the difference between the 75th and 25th percentile of the data. [Q1−1.5×IQR, Q3+1.5×IQR], where Q1 — (25%), а Q3 — (75%).

Why IQR: IQR is a robust method for detecting outliers because it does not depend on the mean and standard deviation, making it more robust to extreme values ​​in the data.

○ How did you address inconsistencies in categorical data?

This is done by converting text to a uniform format using the str.lower() method (converting to lower case) or str.title() (format with the first letter of each word capitalized). Converting text to lower case eliminates spelling differences. Converting to a capitalized format helps standardize names such as cities.

# Exercise 6: Splitting Data into Training and Testing Sets

1. **Objective**: Prepare the data for model training by splitting it into training and testing sets.
2. **Steps**:
   1. Use sklearn.model\_selection.train\_test\_split() to split the dataset

into training and testing sets.

○ Ensure that the target variable is correctly separated from the features.

○ Explore the impact of different train-test split ratios (e.g., 70-30, 80-20) on model performance.

1. **Questions**:
   1. How do you split a dataset into training and testing sets in Python?

To split the dataset into training and test sets, the train\_test\_split() function from the sklearn.model\_selection library is used.

○ What considerations should you keep in mind when choosing a train-test split ratio?

Data size: If you have a small data set, it may be useful to leave more data for training the model so that it can adapt better. In this case, a ratio of 80-20 or even 90-10 is often chosen.

Problem complexity: The more complex the problem, the more data is needed to train the model so that it can detect complex patterns. For example, a ratio of 70-30 may be used for complex problems.

Model generalization ability: To test how well the model can generalize to new data, the test set should be large enough to represent all possible situations.

Overfitting: If the test set is too small, you may overestimate the generalization ability of the model. Too much training data can lead to overfitting.

○ How does the size of the training set impact the model's ability to generalize?

**Small training set:** If the training set is too small, the model may not learn enough to learn patterns in the data. This can lead to underfitting, where the model predicts poorly on both the training and test data.

**Too large a training set:** If the training set is too large compared to the test set, there is a risk that the model will adapt too well to the training data but will not generalize well to new data. This can lead to overfitting.

**Optimal training set size:** To achieve good generalization, a balance must be found. Typically, a ratio of 70-30 or 80-20 works well, giving the model enough data to train while still leaving enough data to test its generalization ability.

# Exercise 7: Data Preprocessing Pipeline

1. **Objective**: Build a preprocessing pipeline to automate the data preparation process.
2. **Steps**:
   1. Use sklearn.pipeline.Pipeline to create a pipeline that includes steps such as missing value imputation, feature scaling, and encoding categorical variables.

○ Fit the pipeline to the training data and transform the test data.

○ Integrate the preprocessing pipeline with a machine learning model for end-to-end training and evaluation.

1. **Questions**:
   1. What are the benefits of using a preprocessing pipeline?

Preprocessing pipelines simplify code, automate the process of applying transformations, and ensure reproducibility of results. They help prevent errors and data leakage by combining all processing steps into a single object.

○ How does the pipeline ensure consistency between training and test data transformations?

A pipeline ensures consistency by applying the same steps and parameters to training and test data, since all transformations are performed consistently in a single object, eliminating inconsistencies and information leakage.

○ How can you extend the pipeline to include additional preprocessing steps?

You can extend a pipeline by adding new steps, such as additional imputation methods, scaling, polynomial features, or date-based transformations, by integrating them into the pipeline as new data processing steps.