# 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 bfill (backward fill) strategy, which replaces missing values ​​with the next non-zero value in the column. I chose this strategy because it preserves the order of the data and can be useful when the next values ​​are more relevant than the previous ones. This is useful in data where order matters.

○ How did filling missing values affect the dataset?

Filling missing values ​​with the bfill method avoids data loss that would otherwise occur if rows with gaps were removed. This helps to preserve all rows in the dataset, which may be important for further analysis. However, it also introduces an assumption that the next value is meaningful to fill the gap, which may not always be true.

○ 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 if the number of missing values ​​is large or if filling in would distort the data (for example, if the missing values ​​contain important information). It is also appropriate in cases where the data set is large enough that removing a few rows will not significantly affect the results of the analysis.

# 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 and standardization are both techniques for scaling numerical data, but they work in different ways. **Normalization** (also called Min-Max scaling) adjusts the values in the dataset so that they fall within a specific range, typically between 0 and 1.

**Standardization**, on the other hand, transforms data to have a mean of 0 and a standard deviation of 1. This technique is useful when the data distribution matters or for models that assume normal (Gaussian) distribution of the data, such as in logistic regression or support vector machines. Standardization also handles outliers better than normalization because it doesn’t scale the data into a fixed range.

○ How does one-hot encoding transform categorical variables?

**One-hot encoding** transforms categorical variables into multiple binary columns, with each column representing one unique category from the original feature. For example, if a categorical variable has values like ['A', 'B', 'C'], one-hot encoding will create three new binary columns. Each column corresponds to a category (A, B, C), and for every row in the dataset, the column corresponding to that row’s category will have a value of 1, while the other columns will have 0.

For instance, the category 'A' will be encoded as [1, 0, 0], 'B' as [0, 1, 0], and 'C' as [0, 0, 1]. This transformation allows machine learning models, which typically require numeric input, to handle categorical data without assuming an ordinal relationship between categories.

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

Binning continuous variables into categories is often useful when you want to **simplify** the data or when exact numerical values are not as important as the range or category they fall into. For example, instead of using a person's exact age, you might group people into categories like "young", "middle-aged", and "senior".

Binning can also be helpful in **reducing the effect of outliers**, as extreme values get grouped into broader categories. This method can also aid in **pattern recognition**, making it easier to identify trends in specific groups rather than working with individual continuous values. It’s a great way to make data more interpretable and manageable for certain types of analysis or decision-making processes.

# 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?

I identified duplicate rows using the duplicated() method, which flags rows as True if they have the same values across all columns as a previous row. To remove these duplicate rows, I used the drop\_duplicates() method, which keeps the first occurrence of each duplicate row and removes subsequent duplicates. This method ensures that only unique rows remain, cleaning the dataset of redundancy.

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

To detect and remove outliers, I used the **Interquartile Range (IQR)** method. IQR is the range between the 1st quartile (Q1) and the 3rd quartile (Q3) of the data, which captures the middle 50% of the data. I calculated the IQR and then defined the lower and upper bounds as Q1 - 1.5 \* IQR and Q3 + 1.5 \* IQR, respectively. Any value outside these bounds was considered an outlier and removed. The IQR method is effective for skewed data or data with extreme values, as it is less affected by outliers compared to other methods like Z-score.

○ How did you address inconsistencies in categorical data?

# To address inconsistencies in categorical data, I used string manipulation methods such as str.lower() to convert all text to lowercase, ensuring uniformity. I then applied str.capitalize() or str.title() to standardize the format (e.g., making the first letter uppercase and the rest lowercase for names and categories). This method is useful when data contains variations in case (e.g., "sales" vs "Sales") or different spellings of the same category, helping maintain consistency across categorical variables.

# 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?

In Python, you can split a dataset into training and testing sets using the train\_test\_split() function from sklearn.model\_selection. This function randomly divides your dataset into two parts: one used for training the model and the other for testing its performance.   
from sklearn.model\_selection import train\_test\_split # Assuming 'X' is your feature set and 'y' is the target variable X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

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

When choosing a train-test split ratio, you should consider the following:

1. **Size of the dataset**: For smaller datasets, you might want to allocate a larger proportion of the data for training to ensure the model learns effectively. Common split ratios include:
   * 70-30 (70% training, 30% testing)
   * 80-20 (80% training, 20% testing)
2. **Balance of classes**: If your dataset has imbalanced classes, ensure that both the training and test sets maintain a similar class distribution. Stratified sampling (using stratify=y in train\_test\_split) can help maintain the class proportions.
3. **Generalization**: The testing set should be large enough to give a reliable estimate of the model's performance on unseen data. However, the training set should still be large enough to allow the model to learn meaningful patterns.
4. **Cross-validation**: Sometimes, rather than relying on a single train-test split, you can use cross-validation, which involves multiple train-test splits to give a better idea of model performance.

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

The size of the training set plays a crucial role in how well the model generalizes:

*Too small a training set*: If the training set is too small, the model may not have enough data to learn patterns, resulting in underfitting. The model might not capture the complexity of the data, leading to poor performance on both the training and test sets.

*Too large a training set*: While a larger training set generally improves the model's ability to learn, if the test set is too small, you might not have a reliable estimate of how well the model will perform on unseen data.

*Finding a balance*: The goal is to find a balance where the training set is large enough to help the model learn effectively, while the test set is sufficiently large to provide a reliable estimate of model performance. Larger datasets typically benefit from smaller test sets (e.g., 90-10 split), while smaller datasets might need a larger test set for reliable evaluation.

# 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?

Using a preprocessing pipeline consolidates multiple preprocessing tasks into a single workflow, automating steps like missing value imputation, feature scaling, and encoding. This streamlines the process, ensures consistency across different stages of model training and evaluation, and reduces code redundancy. It also simplifies maintenance and experimentation by allowing easy adjustments to preprocessing steps and model configurations.

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

The pipeline ensures consistency by fitting the preprocessing steps only on the training data, such as calculating scaling factors or imputing values. This ensures that transformations are based on the training set and then consistently applied to both training and test data, preventing data leakage and ensuring that the model is evaluated on data transformed in the same way it was trained.

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

To extend the pipeline, additional preprocessing steps can be added by chaining them in the pipeline configuration. For instance, you can include feature engineering steps like polynomial feature generation, or custom transformations tailored to specific needs. This modular approach allows for flexible and scalable preprocessing adjustments as needed for different datasets or problems.