# 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?

You can load a CSV file into a DataFrame by using the pd.read\_csv('file\_name.csv') function, where file\_name.csv is the name of the file you want to load.

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

The info() function gives an overview of the dataset, including the number of rows and columns, the column names, the data types of each column, the number of non-missing values in each column, and the total memory usage of the DataFrame..

○ How can you identify missing values in the dataset?

Missing values can be detected using the isnull() function. This function returns a DataFrame where each entry is marked as True if it's missing, and False if it's not. To get the total number of missing values per column, you can use 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 chose to fill missing values with a fixed date ('2024-01-01'). This method was selected to retain all data rows while ensuring consistency. This approach helps avoid data loss and makes sense contextually, especially if the missing values represent something important, like a specific event or a cutoff point..

○ How did filling missing values affect the dataset?

Replacing missing values ensures that all data rows are complete and usable. However, filling missing data with a set value can introduce bias, especially if the chosen value doesn't reflect the actual distribution of the data.

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

Dropping rows with missing data may be preferable when the missing data is extensive or essential to the analysis, and filling them could distort the results. Also, when the dataset has enough non-missing rows, removing incomplete rows might improve data quality without compromising accuracy.

# 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 rescales the data to a [0, 1] range, useful for aligning variables on different scales. Standardization adjusts data to have a mean of 0 and a standard deviation of 1, which is useful for normally distributed data or when centering around the mean is essential.

.

○ How does one-hot encoding transform categorical variables?

One-hot encoding converts categorical data into binary columns, where each unique category becomes a separate column, represented as 0 or 1. This allows machine learning algorithms to process categorical data as numeric inputs.

.

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

Binning simplifies continuous data by grouping it into intervals, which can make it easier to analyze and interpret, especially when identifying trends or patterns. It’s useful when working with models that benefit from categorized data..

# 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?

I generated polynomial features such as the squares of two variables (Feature1^2, Feature2^2) and their interaction term (Feature1 \* Feature2). These features were introduced to allow the model to capture more complex relationships between variables, which could enhance predictive power

○ How did the new features improve the dataset?

These features expanded the feature space, allowing the model to recognize nonlinear relationships, potentially improving accuracy. However, adding too many features may increase the risk of overfitting if the data sample is too small.

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

Date-based features (e.g., year, month, weekday) can help capture temporal patterns, which are valuable for tasks like sales forecasting or trend analysis. Features like "time since event" can reveal long-term trends, while day-of-week or quarter-based features are often useful in specific domains like finance or retail.

# 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 used the df.duplicated() method to detect duplicate rows and then applied drop\_duplicates() to remove any repeating records, ensuring each row was unique.

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

I used the Interquartile Range (IQR) method to detect outliers. IQR is less sensitive to extreme values and is a robust method for identifying unusual data points by focusing on the middle 50% of the dataset.

○ How did you address inconsistencies in categorical data?

I standardized categorical data by using text formatting functions like str.lower() to convert all entries to lowercase, ensuring consistency in how categories are labeled.

# 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?

The train\_test\_split() function from sklearn.model\_selection is used to split the dataset into training and testing sets, ensuring that the target variable is separated from the features.

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

Consider the size of your dataset, the complexity of the problem, and the need for the model to generalize well. A common split ratio is 80-20, but for small datasets, you might choose 90-10 to ensure the model has enough training data.

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

If the training set is too small, the model may not learn enough to recognize patterns, leading to underfitting. If the training set is too large, the model may overfit and perform poorly on new, unseen data. A balanced split helps avoid these issues.

# 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?

A pipeline automates the data transformation process, ensuring that all preprocessing steps are applied consistently and efficiently. It reduces the risk of human error and helps maintain reproducibility

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

Since the pipeline applies the same steps and transformations to both the training and test datasets, it prevents discrepancies and ensures the model sees the data in a uniform format during both training and testing.

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

Additional steps like outlier detection, more advanced feature engineering, or different scaling techniques can be added to the pipeline, allowing for a more comprehensive data preparation process.