**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-ACLGdHYZq1gVZbUeBzIg/viewform?usp=sf_link>

### **Exercise 1: Loading Data with Pandas**

1. **Objective**: Learn how to load and inspect datasets using Pandas.
2. **Steps**:
   * 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.
3. **Questions**:
   * How do you load a CSV file into a Pandas DataFrame?  
     To load a CSV file into a Pandas DataFrame, the read\_csv() function is used.
   * What information does the info() function provide about the dataset?  
     The info() function in Pandas provides an overview of the dataset, showing the number of entries, the number of columns, and the data types for each column (e.g., integers, floats, or strings). It also displays how many values are non-null in each column, giving insight into whether the dataset contains any missing values. This function is useful for quickly understanding the structure of the dataset and its completeness.
   * How can you identify missing values in the dataset?  
     Missing values in a dataset can be identified by using methods that check for gaps in the data. In Pandas, missing values are typically represented as `NaN` (Not a Number). `True` indicates the presence of a missing value in each cell of the DataFrame. 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**:
   * 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.
3. **Questions**:
   * 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 median for numerical data, which was 45). This approach was chosen to maintain the structure of the dataset and ensure that gaps were filled with values that made sense in the context of the data. By using the median value of 45, I minimized the risk of skewing the data, especially when the missing values affected around 15% of the dataset. This method helps to avoid information loss while preserving data consistency.
   * How did filling missing values affect the dataset?  
     Filling missing values completed the dataset, allowing for smoother analysis without gaps. However, it may have introduced some bias, as the filled values don't always represent the true distribution of the data. While it helped maintain consistency, there's a risk that the imputed values may not fully capture the underlying variability.
   * When might it be more appropriate to drop rows with missing values instead of filling them?  
     Dropping rows with missing values is more appropriate when the amount of missing data is small or when filling it could distort the analysis. If only a few rows are affected or if the missing data is crucial and cannot be accurately imputed, it's better to remove those rows to preserve the integrity of the dataset and avoid introducing errors or bias.

### **Exercise 3: Data Transformation**

1. **Objective**: Transform data to prepare it for analysis.
2. **Steps**:
   * 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.
3. **Questions**:
   * What is the difference between normalization and standardization?  
     Normalization and standardization are two techniques used to scale data. Normalization rescales the data to a fixed range, typically between 0 and 1, ensuring that all features contribute equally to the model. Standardization, on the other hand, transforms the data to have a mean of 0 and a standard deviation of 1. This technique is useful when data follows a normal distribution and when it's important to center the data for certain algorithms like SVM or logistic regression.
   * How does one-hot encoding transform categorical variables?  
     One-hot encoding transforms categorical variables into a binary format by creating new columns, one for each category. Each column contains 0 or 1 values, indicating the presence or absence of a category in the original variable. This approach allows categorical data to be represented numerically so it can be used in machine learning models without implying any ordinal relationship between the categories.
   * Why might you want to bin continuous variables into categories?  
     Binning continuous variables into categories can simplify the data and make it easier to interpret. This is especially useful when the precise values of a continuous variable are less important than the general range they fall into. Binning can also help reduce the impact of outliers and make models more robust by grouping data into more manageable segments or categories, such as low, medium, and high ranges.

### **Exercise 4: Feature Engineering**

1. **Objective**: Create new features to improve the predictive power of a dataset.
2. **Steps**:
   * 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.
3. **Questions**:
   * What new features did you create, and why?  
     In this case, I created polynomial features based on two existing features, “Feature1” and “Feature2.” These included the squared values of each feature—Feature1² and Feature2²—along with the interaction between them (Feature1 \* Feature2). These new features were introduced to help the model capture non-linear relationships and dependencies between variables, which are often missed by using the raw features alone. For example, combining these features allows the model to detect quadratic or multiplicative effects, which can significantly improve the model’s predictive accuracy. Additionally, I derived date-based features like “day of the week” and “month” from a datetime column, enabling the model to capture time-related patterns, such as increased activity during weekends or holiday months. These new features were designed to provide the model with richer, more nuanced data for better predictions.
   * How did the new features improve the dataset?  
     The new features improved the dataset by adding more complexity and capturing hidden relationships between the variables. By introducing polynomial features like Feature1², Feature2², and their interaction (Feature1 \* Feature2), the dataset became more expressive, allowing the model to better capture non-linear relationships. This is particularly useful when the relationship between variables isn’t purely linear, such as when higher values of one feature impact the outcome differently than lower values.

Additionally, the date-based features helped the model uncover time-based patterns. For example, extracting the “day of the week” and “month” allowed the model to identify seasonal trends or day-specific behaviors, like increased sales on weekends or during holiday periods. These features enriched the dataset with temporal information that was previously unavailable, leading to more accurate predictions and better insights from the data. The combination of these new features allowed the model to better understand the relationships and variations in the data, ultimately improving its performance.

* + How can date-based features be useful in a dataset?  
    Date-based features are incredibly useful in a dataset because they allow the model to capture time-related trends and patterns that are often critical for making accurate predictions. By extracting elements such as the “year,” “month,” “day of the week,” or even “hour of the day,” the dataset becomes enriched with temporal context that can explain fluctuations in behaviors or outcomes over time.

For example, sales or customer activity might spike on weekends or during certain months (e.g., holiday seasons like December). By including features such as “day of the week” or “month,” the model can account for these seasonal or periodic effects. Similarly, features like “time since last purchase” or “days until renewal” can be used to predict customer churn, retention, or upcoming renewals in subscription-based services.

Moreover, date-based features help detect cyclical trends, such as quarterly financial cycles or daily business patterns. This can be critical for forecasting, inventory management, or even staffing decisions, as businesses may adjust their operations according to expected peak periods.

In summary, date-based features offer valuable insights into the temporal dynamics of the data, making them essential for capturing seasonality, time trends, and recurring behaviors that can significantly enhance the predictive power of a model.

### **Exercise 5: Data Cleaning**

1. **Objective**: Clean data to ensure it's ready for analysis.
2. **Steps**:
   * 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).
3. **Questions**:
   * How did you identify and handle duplicate rows in the dataset?  
     To identify duplicate rows, I used a simple method of comparing rows in the dataset. I looked for cases where the entire row of data was identical to another row, which indicates a duplicate. These duplicates could occur due to data collection errors or multiple entries of the same record. Once the duplicates were identified, I removed them using the drop\_duplicates() function. Removing these rows ensured that the analysis would not be skewed by repeated information, which could distort patterns and lead to incorrect conclusions. By cleaning up these duplicates, I made sure that each record in the dataset was unique and contributed meaningful information.
   * What method did you use to detect and remove outliers, and why?  
     To detect outliers, I used the Interquartile Range (IQR) method, a widely-used technique for identifying extreme values in a dataset. This method calculates the range between the 25th percentile (Q1) and the 75th percentile (Q3), which represents the middle 50% of the data. Outliers are identified as values that fall below Q1 - 1.5 \* IQR or above Q3 + 1.5 \* IQR. The advantage of the IQR method is that it doesn't rely on the mean or standard deviation, which makes it more robust and less influenced by extreme values, especially in skewed datasets. After identifying these outliers, I carefully removed them to ensure that they wouldn't distort the analysis or bias the model's predictions. This process helped in cleaning the dataset, ensuring that the remaining data accurately represented the core patterns and trends without being affected by anomalies or erroneous data points.
   * How did you address inconsistencies in categorical data?  
     To address inconsistencies in categorical data, I standardized the format of the text entries by converting them to a uniform case. This was achieved by using the `.lower()` method to convert all text to lowercase, which eliminated any discrepancies caused by different capitalizations. For example, entries like "USA" and "usa" would be treated the same after conversion. Additionally, I applied the `.title()` method where necessary, particularly for proper nouns such as city names or product titles, to ensure consistency in capitalization (e.g., "new york" becomes "New York"). These transformations helped to merge similar categories that may have been recorded differently due to formatting issues, ensuring that the dataset was clean and ready for analysis without redundant variations.

### **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**:
   * 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.
3. **Questions**:
   * How do you split a dataset into training and testing sets in Python?  
     In Python, the train\_test\_split() function from the sklearn.model\_selection module is used to split a dataset into training and testing sets. The function takes the feature matrix (X) and target variable (y) as inputs, along with a parameter for the test size, which determines the proportion of the dataset to be used for testing. For example, if you want a 70-30 split, you would set test\_size=0.30. The function then randomly assigns a portion of the data to the training set and the rest to the testing set, ensuring that the target variable is correctly separated from the features. Additionally, you can set a random seed using the random\_state parameter to ensure reproducibility, which is important for consistent results during model evaluation.
   * What considerations should you keep in mind when choosing a train-test split ratio?  
     When choosing a train-test split ratio, it’s important to consider the size of the dataset and the balance between training and testing data. A typical split ratio is 80-20, meaning 80% of the data is used for training and 20% for testing. If you have a larger dataset, you may use a smaller portion for testing, like 90-10, as the model will still have sufficient data to generalize. For smaller datasets, a 70-30 split might be more appropriate to ensure there is enough data in the training set to build a robust model while keeping a reasonable amount for testing.

It is also crucial to ensure that the split is representative of the entire dataset. For example, if the data is imbalanced (e.g., a classification problem with more instances of one class than another), it’s important to maintain this balance in both the training and testing sets. This can be done using the stratify parameter in train\_test\_split(), which ensures that the distribution of the target variable is similar across the two sets.

* + How does the size of the training set impact the model's ability to generalize?  
    The size of the training set plays a significant role in the model’s ability to generalize to new, unseen data. A larger training set generally allows the model to learn better, as it has more data points to understand patterns and relationships. With more training data, the model can build more accurate representations of the underlying data distribution, which typically leads to better performance on unseen data.

However, if the training set is too small, the model may not capture enough variability in the data, leading to underfitting, where the model performs poorly on both the training and testing sets. On the other hand, if too much data is allocated to the training set and the test set is too small, the model may seem to perform well during testing, but the evaluation might not be reliable due to the limited test data. Therefore, a balance is needed to ensure the model has enough data to learn effectively while reserving enough data for a robust evaluation.

### **Exercise 7: Data Preprocessing Pipeline**

1. **Objective**: Build a preprocessing pipeline to automate the data preparation process.
2. **Steps**:
   * 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.
3. **Questions**:
   * What are the benefits of using a preprocessing pipeline?  
     A preprocessing pipeline automates the data preparation process, reducing errors and ensuring consistent application of steps like imputation, scaling, and encoding. It makes the code more organized and modular, which simplifies maintenance and future modifications. Additionally, pipelines prevent data leakage by applying transformations only after the data is split, ensuring that the training and testing sets are processed identically. Pipelines also integrate smoothly with machine learning models for end-to-end training and evaluation.
   * How does the pipeline ensure consistency between training and test data transformations?  
     Pipelines maintain consistency by calculating transformations (e.g., scaling) based only on the training data and then applying those same transformations to the test data. This prevents data leakage since the test data is not used during the fitting process. The transform() method ensures that the same steps are applied to both sets, making sure they undergo identical processing without refitting on the test data.
   * How can you extend the pipeline to include additional preprocessing steps?  
     Pipelines are flexible and can easily be extended by appending new steps, such as polynomial feature generation, outlier detection, or custom feature engineering. Each step is defined as a name-operation pair, and additional steps can be added or modified as needed. This modularity makes pipelines highly adaptable to changing preprocessing requirements, ensuring scalability and ease of customization for complex workflows.