Machine Learning Lab (AI332L)

Class:	BS Artificial Intelligence	**Semester:**	5th (Fall 2023)
Instructor:	Dr. M. Ehatisham-ul-Haq	**Email:**	ehtisham@mail.au.edu.pk

| Lab 03 |

Data Preprocessing Techniques for Machine Learning in Python

Lab Objective:

In this lab tutorial, you will learn some fundamental data preprocessing techniques in Python for preparing data for machine learning models, which is crucial for achieving reliable results in machine learning tasks.

Setting up Python with Google Colab

1. Go to Google Colab.
2. Create a new notebook by clicking on **File** > **New Notebook**.
3. You're now ready to start writing and executing Python code in the notebook!

Basic Data Preprocessing Techniques in Python

A. Handling Missing Values: Imputation

Imputation is a technique used to fill in missing values in a dataset. When working with real-world data, it's common to encounter missing information for various reasons, such as data collection errors, survey non-response, or technical issues. Imputation helps address these missing values by estimating or replacing them using other information available in the dataset. The goal is to maintain the integrity and usefulness of the dataset for analysis or modeling.

Imputation with Mean/Mode/Median

```
import pandas as pd

# Create a sample dataset with missing values
data = {'A': [1, 2, np.nan, 4, 5],
        'B': [np.nan, 2, 3, np.nan, 5],
        'C': [1, 2, 3, 4, 5]}

df = pd.DataFrame(data)

# Fill missing values with mean of respective columns
df_filled_mean = df.fillna(df.mean())

# Fill missing values with median of respective columns
df_filled_median = df.fillna(df.median())
```

```
# Fill missing values with mode of respective columns
df_filled_mode = df.fillna(df.mode().iloc[0])

print("Filled with Mean:")
print(df_filled_mean)

print("\nFilled with Median:")
print(df_filled_median)

print("\nFilled with Mode:")
print(df_filled_mode)
```

- For numerical variables, missing values can be replaced with the mean (average) or median (middle value) of the known values in the same column.
- For categorical/ordinal variables, the mode (most frequently occurring category) is used.

Imputation with Advanced Techniques (i.e., KNN)

K-Nearest Neighbors (KNN) imputation is a more sophisticated technique. It estimates missing values based on the values of their nearest neighbors in the feature space.

```
from sklearn.impute import KNNImputer
import pandas as pd

# Create a sample dataset with missing values
data = {'A': [1, 2, np.nan, 4, 5],
        'B': [np.nan, 2, 3, np.nan, 5],
        'C': [1, 2, 3, 4, 5]}

df = pd.DataFrame(data)

# Initialize the K-Nearest Neighbors imputer
imputer = KNNImputer(n_neighbors=2)

# Apply KNN imputation
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

print(df_imputed)
```

The choice of imputation method can impact the results of any analysis or modeling performed on the dataset, so it is important to carefully consider the implications of imputation for the specific research or analysis being conducted.

B. Handling Categorical Values: One-Hot Encoding

One-hot encoding is a technique used to convert categorical variables into a format that can be provided to machine learning algorithms to improve predictions. It is particularly useful when dealing with nominal categorical data, where no inherent ordinal relationship exists between the categories.

Example: Let us consider a dataset of fruits with a categorical variable "Color" that can take four values: "Red", "Green", "Yellow", and "Blue".

```
import pandas as pd

data = {'Fruit': ['Apple', 'Banana', 'Cherry', 'Apple', 'Cherry'],
        'Color': ['Red', 'Yellow', 'Red', 'Green', 'Blue']}

df = pd.DataFrame(data)
```

Before One-hot Encoding:

	Fruit	Color
0	Apple	Red
1	Banana	Yellow
2	Cherry	Red
3	Apple	Green
4	Cherry	Blue

Applying One-hot Encoding:

```
df_encoded = pd.get_dummies(df, columns=['Color'], drop_first=True)
```

After One-hot Encoding:

	Fruit	Color_Green	Color_Red	Color_Yellow
0	Apple	0	1	0
1	Banana	0	0	1
2	Cherry	0	1	0
3	Apple	1	0	0
4	Cherry	0	0	0

How Does One-hot Encoding Work?

1. **Identify Categorical Variables**: First, identify which variables are categorical. These are the ones that have discrete values and no intrinsic numerical meaning.
2. **Create Binary Columns**: For each category in the original variable, create a new binary column. In the example above, we created three new columns: Color_Green, Color_Red, and Color_Yellow.
3. **Assign Binary Values**: For each row, assign a value of 1 in the respective binary column if the category is present, and 0 if it is not. This creates a binary representation of the categories.
4. **Drop One Column**: To avoid multicollinearity (where one category can be perfectly predicted from the others), we typically drop one of the binary columns. In the example, we dropped Color_Blue because it can be inferred from the other three columns.

What will happen if you set `drop_first=False` when applying one-hot encoding?

Advantages of One-Hot Encoding:

It preserves categorical information and the most machine learning algorithms work with numerical data, so one-hot encoding makes categorical data compatible with these algorithms.

Considerations:

1. **High Dimensionality**: One-hot encoding can increase the dimensionality of your dataset, especially if you have a large number of categories. This can be a consideration for memory and computation efficiency.
2. **Sparse Matrices**: The resulting matrix can be sparse, which means it contains mostly zero values. Some algorithms are optimized for sparse data, while others are not.
3. **Interpretability**: After one-hot encoding, the interpretation of coefficients in linear models becomes more complex. Each category gets its own coefficient, which can make the model harder to interpret.

After applying one-hot encoding to the categorical variables, the modified dataset can be used for building and training machine learning models in the same way as you would with any other dataset (refer to the Lab 02 for details).

C. Handling Ordinal Values: Label Encoding and Custom Mapping

Handling ordinal values in Python for machine learning tasks involves converting these values into a numerical format that preserves the inherent order or ranking. This can be done using techniques like label encoding or custom mapping.

Method 1: Label Encoding

Label Encoding assigns a unique integer to each category in an ordinal variable. The integers are assigned based on the order or ranking of the categories.

Example: Suppose we have an ordinal variable 'Education_Level' with categories: 'High School', 'Associate Degree', 'Bachelor Degree', 'Master Degree', and 'PhD'. The ranking from lowest to highest is: 'High School' < 'Associate Degree' < 'Bachelor Degree' < 'Master Degree' < 'PhD'.

```
import pandas as pd

# Create a sample dataset
data = pd.DataFrame({
    'Name': ['John', 'Jane', 'Jim', 'Jill', 'Jack'],
    'Education_Level': ['Bachelor Degree', 'High School', 'Master Degree',
'PhD', 'Associate Degree']
})

# Define mapping for ordinal variable
education_mapping = {
    'High School': 1,
    'Associate Degree': 2,
    'Bachelor Degree': 3,
    'Master Degree': 4,
```

```
    'PhD': 5
}

# Apply label encoding to 'Education_Level'
data['Education_Level_Encoded'] =
data['Education_Level'].map(education_mapping)

# Display the modified dataset
print(data)
```

Output after Label Encoding:

Name	Education_Level	Education_Level_Encoded
John	Bachelor's Degree	3
Jane	High School	1
Jim	Master's Degree	4
Jill	PhD	5
Jack	Associate's Degree	2

Method 2: Custom Mapping

In some cases, you may have specific knowledge about the ordinal variable that doesn't follow a natural numerical order. In such cases, you can define a custom mapping.

Example: Suppose we have an ordinal variable 'Satisfaction_Level' with categories: 'Poor', 'Average', 'Good', and 'Excellent'. We'll assign custom values based on our specific knowledge.

```
import pandas as pd

# Create a sample dataset
data = pd.DataFrame({
    'Name': ['John', 'Jane', 'Jim', 'Jill', 'Jack'],
    'Satisfaction_Level': ['Good', 'Average', 'Excellent', 'Poor', 'Good']
})

# Define a custom mapping for ordinal variable
satisfaction_mapping = {
    'Poor': 1,
    'Average': 2,
    'Good': 3,
    'Excellent': 4
}

# Apply custom mapping to 'Satisfaction_Level'
data['Satisfaction_Level_Encoded'] =
data['Satisfaction_Level'].map(satisfaction_mapping)

# Display the modified dataset
print(data)
```

Output after Custom Mapping:

Name	Satisfaction_Level	Satisfaction_Level_Encoded
John	Good	3
Jane	Average	2
Jim	Excellent	4
Jill	Poor	1
Jack	Good	3

D. Feature Scaling with Mutliple Input Variables

Feature scaling in machine learning is one of the most critical steps during the pre-processing of data before creating a machine learning model. Scaling can make a difference between a weak machine learning model and a better one. The most common techniques of feature scaling are Normalization and Standardization.

POINTS TO PONDER:

What is the difference between Normalization and Standardization?

Applying Normalization:

```
from sklearn.preprocessing import MinMaxScaler
# Create a MinMaxScaler instance
scaler = MinMaxScaler()

# Apply normalization to the features
X_normalized = scaler.fit_transform(X)

# Continue with the rest of the code (e.g., splitting data, building models, etc.)
```

Applying Standardization:

```
from sklearn.preprocessing import StandardScaler

# Create a StandardScaler instance
scaler = StandardScaler()

# Apply standardization to the features
X_standardized = scaler.fit_transform(X)

# Continue with the rest of the code (e.g., splitting data, building models, etc.)
```

Lab Task:

Take an example dataset of your own choice, which should have at least four input variables to predict the output value. The feature x_1 should be directly related to the output y, and the other feature x_2 should have a negative relationship with y. Also, the numerical scales x_1 and x_2 should be highly different. The feature x_3 should be a categorical feature, whereas x_4 must be an ordinal feature. Other features (if they exist) can be of any type. The dataset must contain the missing values as well (if there is no missing value, you can intentionally remove some values for each column).

a) Check for the missing values in the dataset, if any, and handle them appropriately using the relevant method.
b) Encode the categorical/ordinal variables using relevant encoding techniques.
c) Find and plot the correlation between different variables. How would you know that a feature is directly or indirectly related to the output variable?
d) Apply the feature scaling techniques on the input variables and re-compute the correlation.
