#1 Data Handling and Preprocessing Lab

- **Objective: To learn how to handle and preprocess datasets for machine learning tasks.**
- **Activities:**
  - **Importing datasets using libraries like pandas in Python.**
  - **Handling missing values: Imputation, removal, or substitution.**
  - **Data scaling and normalization.**
  - **Encoding categorical variables.**
  - **Splitting datasets into training, testing, and validation sets.**

Data Handling and Preprocessing Essentials:

1. **Data Loading: Importing dataset from files like CSV or Excel.**
2. **Data Exploration: Understanding data structure, summary statistics, and identifying missing values.**
3. **Data Visualization: Using graphs to visualize distributions and relationships.**
4. **Missing Values Handling: Removing or imputing missing values.**
5. **Data Encoding: Converting categorical data into numerical format.**
6. **Data Scaling: Ensuring features are on the same scale for fair comparison.**
7. **Data Splitting: Dividing data into training, testing, and validation sets.**
8. **Importance: Enhances model performance, reduces overfitting, and ensures model compatibility.**
9. **Best Practices: Conducting exploratory data analysis, building preprocessing pipelines, and updating processes iteratively.**

1. Loading the Dataset:

```
import pandas as pd

# Load the dataset (assuming 'data.csv' is the dataset file)

data = pd.read_csv('data.csv')
```

2. Data Exploration:

```
# Display the first few rows of the dataset

print("First few rows of the dataset:")

print(data.head())

# Display summary statistics of the dataset

print("\nSummary statistics of the dataset:")

print(data.describe())

# Check for missing values
```

```
print("\nMissing values in the dataset:")

print(data.isnull().sum())

# Display datatype and non-null counts for each column

print("\nDatatype and non-null counts of each column:")

print(data.info())
```

3. Handling Missing Values:

```
# Option 1: Remove rows with missing values

# data = data.dropna()

# Option 2: Impute missing values (e.g., using mean or median)

# data['column_name'].fillna(data['column_name'].mean(), inplace=True)
```

4. Data Encoding (for Categorical Variables):

```
from sklearn.preprocessing import LabelEncoder

# Assuming 'category_column' is a categorical variable

label_encoder = LabelEncoder()

data['category_column'] = label_encoder.fit_transform(data['category_column'])
```

5. Data Scaling and Normalization (for Numerical Variables):

```
from sklearn.preprocessing import StandardScaler

# Assuming 'numeric_columns' are numerical variables

scaler = StandardScaler()

data['numeric_columns'] = scaler.fit_transform(data['numeric_columns'])

from sklearn.preprocessing import StandardScaler

# Assuming 'numeric_columns' are numerical variables

scaler = StandardScaler()

data['numeric_columns'] = scaler.fit_transform(data['numeric_columns'])
```

6. Splitting the Dataset:

```
from sklearn.model_selection import train_test_split

# Assuming 'target_column' is the target variable

X = data.drop('target_column', axis=1)

y = data['target_column']

# Splitting the dataset into training, testing, and validation sets

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)

X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

7. Final Dataset Information:

```
# Display final dataset information after preprocessing

print("\nFinal dataset information after preprocessing:")

print(data.info())

# Save the preprocessed dataset if needed

# data.to_csv('preprocessed_data.csv', index=False)
```

#2. Supervised Learning Lab

- **Objective: To implement and evaluate supervised learning algorithms.**
- **Activities:**
  - **Implementing algorithms such as K-Nearest Neighbors, Linear Regression, Logistic Regression, and Support Vector Machine using scikit-learn.**
  - **Evaluating models using performance metrics like accuracy, precision, recall, and F1-score.**
  - **Visualizing decision boundaries and model performance using plots and graphs.**
  - **Tuning hyperparameters using techniques like grid search or random search.**

1. Introduction to Supervised Learning:

- **Definition: Supervised learning involves training a model on labeled data, where each example is paired with a corresponding target label.**
- **Objective: Predicting the target label for new, unseen data based on patterns learned from the training data.**

2. Common Supervised Learning Algorithms:

- **Linear Regression: Used for predicting continuous target variables based on linear relationships between features.**
- **Logistic Regression: Suitable for binary classification tasks, predicting probabilities of class membership.**
- **K-Nearest Neighbors (KNN): Non-parametric algorithm for classification and regression based on similarity to nearby data points.**
- **Support Vector Machine (SVM): Effective for both classification and regression tasks, creating decision boundaries to separate classes or predict values.**

3. Model Evaluation Metrics:

- **Regression Metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), R-squared (R2).**
- **Classification Metrics: Confusion Matrix, Accuracy, Precision, Recall, F1-Score, ROC Curve, and AUC (Area Under Curve).**

Code Implementation:

Linear Regression:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

# Data

x = np.array([[15], [23], [18], [23], [24], [22], [22], [19], [19], [16], [24], [11], [24], [16], [23]])

y = np.array([49, 63, 68, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68])

# Linear regression model

model = LinearRegression()

model.fit(x, y)

# Predict for a new value

new_value = np.array([[20]])

predicted_value = model.predict(new_value)

print(model.coef_)

print(model.intercept_)

print("Predicted value for 20:", predicted_value)
```

```
# Plotting the data and regression line

plt.scatter(x, y, color='blue', label='Data Points')

plt.plot(x, model.predict(x), color='red', label='Regression Line')

plt.scatter(new_value, predicted_value, color='green', label='Predicted Value (20)')

plt.xlabel('X')

plt.ylabel('Y')

plt.title('Linear Regression')

plt.legend()

plt.grid(True)

plt.show()
```

#3. Unsupervised Learning Lab

Objective: To implement and evaluate unsupervised learning algorithms.

Activities:

Implementing clustering algorithms such as K-means clustering and hierarchical clustering.

Visualizing clusters using scatter plots or dendrograms.

Evaluating clustering performance using metrics like silhouette score or inertia.

Applying dimensionality reduction techniques like PCA (Principal Component Analysis) and visualizing the reduced feature space.

1. Introduction to Unsupervised Learning:

- **Definition: Unsupervised learning involves training models on unlabeled data to discover patterns or structures within the data.**
- **Objective: Extract meaningful insights or groupings from the data without explicit supervision.**

2. Common Unsupervised Learning Algorithms:

- **Clustering Algorithms:**
  - **K-Means Clustering: Partitioning the data into K clusters based on similarity.**

    - **Hierarchical Clustering: Agglomerative or divisive approach to create a tree-like structure of clusters.**
- **Dimensionality Reduction Techniques:**
    - **Principal Component Analysis (PCA): Reducing the dimensionality of the data while preserving variance.**
    - **t-Distributed Stochastic Neighbor Embedding (t-SNE): Non-linear technique for visualization of high-dimensional data.**

3. Visualizing Clusters:

- **Scatter Plots: Visualizing clusters in 2D or 3D space using scatter plots.**
- **Dendrograms: Visualizing hierarchical clustering results as dendrograms to show the hierarchy of clusters.**

4. Evaluating Clustering Performance:

- **Silhouette Score: Metric to assess the quality of clusters based on the average distance between clusters and within clusters.**
- **Inertia: Sum of squared distances of samples to their closest cluster center, used for evaluating K-Means clustering.**

Code Implementation:

K-Means Clustering:

```
from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Assuming X is the dataset

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

labels = kmeans.labels_

# Visualizing clusters

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('K-Means Clustering')

plt.show()
```

```
# Evaluating clustering performance (Inertia)

inertia = kmeans.inertia_

print("Inertia:", inertia)
```

4#. Deep Learning Lab

Objective: To introduce basic concepts of deep learning and practical implementation.

Activities:

Building and training a simple neural network using libraries like TensorFlow or PyTorch.

Implementing common architectures like feedforward neural networks or convolutional neural networks (CNNs).

Training models for tasks like image classification using datasets such as MNIST or CIFAR-10.

Visualizing learned features using techniques like activation maximization or t-SNE.

1. Introduction to Deep Learning:

- **Definition: Deep learning is a subset of machine learning that utilizes neural networks with multiple layers to learn hierarchical representations of data.**
- **Objective: Understand the fundamental concepts of deep learning and its applications in various domains like image recognition, natural language processing, and more.**

2. Building and Training Neural Networks:

- **Neural Network Architecture: Designing the structure of a neural network, including the number of layers, types of layers (e.g., dense, convolutional, recurrent), and activation functions.**
- **Training Process: Iteratively adjusting the model parameters (weights and biases) using optimization algorithms like gradient descent to minimize the loss function.**
- **Backpropagation: The process of computing gradients of the loss function with respect to the model parameters, which allows for efficient parameter updates during training.**

3. Implementing Common Architectures:

- **Feedforward Neural Networks (FNNs): Basic neural network architecture where information flows in one direction from input to output.**
- **Convolutional Neural Networks (CNNs): Specialized neural network architecture for processing grid-like data, such as images, by using convolutional layers to extract spatial hierarchies of features.**

4. Training Models for Image Classification:

- **Datasets: Utilizing standard datasets like MNIST (handwritten digits) or CIFAR-10 (small images of 10 different classes) for image classification tasks.**
- **Model Training: Training neural network models to classify images into predefined categories (e.g., digits from 0 to 9 in the case of MNIST).**

5. Visualizing Learned Features:

- **Activation Maximization: Generating input patterns that maximize the activation of specific neurons in the network to visualize what the network has learned.**
- **t-SNE (t-Distributed Stochastic Neighbor Embedding): Dimensionality reduction technique for visualizing high-dimensional data by preserving local structures.**

Python Code Implementation:

Building and Training a Simple Neural Network (using TensorFlow):

```
import tensorflow as tf

# Define the model architecture

model = tf.keras.Sequential([

  tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),

  tf.keras.layers.Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
```

#5. Natural Language Processing (NLP) and Computer Vision Lab

Objective: To apply machine learning techniques to text and image data.

Activities:

Implementing NLP tasks like text classification or sentiment analysis using techniques like TF-IDF or word embeddings.

Training models for image recognition tasks using pre-trained models like VGG or ResNet.

Fine-tuning pre-trained models on custom datasets for specific tasks.

Visualizing model predictions and understanding model behavior on text and image data.

1. Introduction to Natural Language Processing (NLP):

- **Definition: NLP is a field of artificial intelligence that focuses on the interaction between computers and humans through natural language.**
- **Objective: Understand and manipulate human language text data for various tasks such as classification, sentiment analysis, and machine translation.**

2. Common NLP Tasks:

- **Text Classification: Assigning predefined categories or labels to text documents based on their content.**
- **Sentiment Analysis: Determining the sentiment expressed in a piece of text (positive, negative, or neutral).**
- **Word Embeddings: Representing words as dense vectors in a continuous vector space, capturing semantic similarities between words.**

3. Implementing NLP Tasks:

- **TF-IDF (Term Frequency-Inverse Document Frequency): Technique for text representation that assigns weights to words based on their frequency in a document and inverse frequency across all documents.**
- **Word Embeddings (e.g., Word2Vec, GloVe): Learning dense vector representations of words from large text corpora, capturing semantic relationships between words.**

4. Introduction to Computer Vision:

- **Definition: Computer vision is a field of artificial intelligence that enables computers to interpret and understand the visual world.**
- **Objective: Extract meaningful information from visual data such as images or videos for tasks like image classification, object detection, and image segmentation.**

5. Common Computer Vision Tasks:

- **Image Recognition: Identifying objects or patterns in images and classifying them into predefined categories.**
- **Pre-trained Models: Leveraging pre-trained deep learning models like VGG, ResNet, or Inception for image recognition tasks.**
- **Fine-tuning: Adapting pre-trained models on custom datasets by fine-tuning their parameters to improve performance on specific tasks.**

6. Implementing Computer Vision Tasks:

- **Loading Pre-trained Models: Loading pre-trained models using deep learning frameworks like TensorFlow or PyTorch.**
- **Model Evaluation: Evaluating model performance on test datasets using metrics like accuracy, precision, and recall.**
- **Visualizing Predictions: Visualizing model predictions and understanding model behavior using techniques like confusion matrices and class activation maps.**

Python Code Implementation:

Implementing NLP Tasks:

```
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

# Assuming 'X' is a list of text documents and 'y' is corresponding labels

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# TF-IDF Vectorization

tfidf_vectorizer = TfidfVectorizer(max_features=1000)

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Training a Logistic Regression classifier

clf = LogisticRegression()

clf.fit(X_train_tfidf, y_train)

# Evaluating model performance

y_pred = clf.predict(X_test_tfidf)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```