## Lab – 01: Introduction to Machine Learning and Python Environment

**Aim:** To get familiar with machine learning concepts and set up the Python environment for ML experiments.

### Experiments

1. **Setting Up Python and Jupyter Notebook**
   - Install Python, Jupyter Notebook, and required libraries (`numpy`, `pandas`, `scikit-learn`, `matplotlib`).
   - Verify installation and create a simple notebook.
2. **Basic Python Operations for ML**
   - Practice Python data types, lists, dictionaries, loops, and functions.
   - Perform simple calculations using `numpy`.
3. **Loading and Exploring Datasets**
   - Load a dataset (e.g., Iris dataset) using `pandas`.
   - Display basic statistics and first few records using `head()`.
4. **Data Visualization**
   - Plot simple graphs using `matplotlib` and `seaborn`.
   - Visualize distributions, scatter plots, and correlations between features.
5. **Understanding Train-Test Split**
   - Split a dataset into training and testing sets using `sklearn.model_selection.train_test_split`.
   - Display shapes of training and testing datasets.

## Lab – 02: Data Preprocessing

**Aim:** To understand and implement techniques for cleaning, transforming, and preparing data for machine learning.

### Experiments

1. **Handling Missing Values**
   - Load a dataset with missing values (e.g., `pandas` DataFrame).
   - Identify missing values using `isnull()` and `sum()`.
   - Handle missing data by **removing rows**, **filling with mean/median/mode**, or **forward/backward filling**.
2. **Encoding Categorical Data**
   - Identify categorical features in the dataset.
   - Apply **Label Encoding** for ordinal data and **One-Hot Encoding** for nominal data using `sklearn.preprocessing` or `pandas.get_dummies()`.

## Lab – 03: Exploratory Data Analysis (EDA)

**Aim:** To explore and analyze datasets to uncover patterns, detect anomalies, and summarize key statistics.

**Experiments**

1. **Loading and Inspecting Data**
   - Load a dataset (e.g., Iris, Titanic) using `pandas`.
   - Inspect data using `head()`, `tail()`, `info()`, `describe()` methods.
2. **Statistical Summary of Data**
   - Calculate mean, median, mode, standard deviation, variance, and correlation for numerical features.
3. **Handling Missing Values and Outliers**
   - Identify missing values and outliers.
   - Visualize outliers using boxplots and handle them appropriately.
4. **Data Visualization**
   - Plot histograms, scatter plots, and bar charts using `matplotlib` or `seaborn`.
   - Visualize relationships between features (e.g., pairplots, correlation heatmaps).
5. **Feature Analysis**
   - Analyze categorical variables using `value_counts()` and bar plots.
   - Examine distribution of numerical features using density plots or boxplots.

## Lab – 04: Linear Regression

**Aim:** To implement and evaluate simple and multiple linear regression models using Python.

**Experiments**

1. **Simple Linear Regression**
   - Load a dataset (e.g., `Salary vs Experience`).
   - Fit a simple linear regression model using `sklearn.linear_model.LinearRegression`.
   - Predict target values and visualize the regression line.
2. **Multiple Linear Regression**
   - Load a dataset with multiple features (e.g., Boston Housing dataset).
   - Fit a multiple linear regression model to predict the target variable.
3. **Evaluating Model Performance**
   - Calculate metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and $R^2$ score.
4. **Visualizing Residuals**
   - Plot residuals (difference between actual and predicted values).
   - Check for patterns to validate model assumptions.
5. **Predicting New Data**
   - Use the trained regression model to predict outcomes for new input data.
   - Compare predicted vs actual values to analyze performance.

## Lab – 05: Logistic Regression

**Aim:** To implement and evaluate logistic regression models for classification tasks using Python.

**Experiments**

1. **Binary Classification using Logistic Regression**
    - Load a dataset (e.g., `Titanic` or `Iris` for binary classes).
    - Fit a logistic regression model using `sklearn.linear_model.LogisticRegression`.
    - Predict target labels and evaluate model accuracy.
2. **Confusion Matrix and Classification Report**
    - Generate a confusion matrix to analyze true positives, true negatives, false positives, and false negatives.
    - Display precision, recall, F1-score using `classification_report`.
3. **ROC Curve and AUC Score**
    - Plot the ROC curve to evaluate model performance.
    - Calculate the Area Under Curve (AUC) score.
4. **Multiclass Classification using Logistic Regression**
    - Apply logistic regression to a multiclass dataset (e.g., Iris dataset with 3 classes).
    - Use `one-vs-rest` or `softmax` strategy for multiclass classification.
5. **Predicting New Data**
    - Use the trained logistic regression model to predict the class of new input samples.
    - Compare predicted vs actual values to assess performance.

## Lab – 06: Decision Trees

**Aim:** To implement and evaluate decision tree models for classification and regression tasks using Python.

**Experiments**

1. **Building a Simple Decision Tree**
    - Load a dataset (e.g., `Iris`).
    - Fit a decision tree classifier using `sklearn.tree.DecisionTreeClassifier`.
    - Visualize the tree structure using `plot_tree` or `export_graphviz`.
2. **Evaluating Decision Tree Performance**
    - Split the dataset into training and testing sets.
    - Calculate accuracy, precision, recall, and F1-score for predictions.
3. **Using Different Splitting Criteria**
    - Fit decision trees using different criteria (`gini`, `entropy`).
    - Compare model performance metrics and tree structures.
4. **Pruning and Controlling Overfitting**
    - Use `max_depth`, `min_samples_split`, and `min_samples_leaf` parameters to prevent overfitting.
    - Observe changes in tree complexity and model accuracy.
5. **Predicting New Data**
    - Use the trained decision tree model to predict classes for new input samples.
    - Compare predicted vs actual values to validate the model.

## Lab – 07: Random Forests and Ensemble Methods

**Aim:** To implement and evaluate ensemble learning techniques using Random Forests for classification and regression.

### Experiments

1. **Building a Random Forest Classifier**
    - Load a dataset (e.g., `Iris` or `Titanic`).
    - Fit a `RandomForestClassifier` using `sklearn.ensemble.RandomForestClassifier`.
    - Predict target labels and evaluate model accuracy.
2. **Feature Importance Analysis**
    - Extract and visualize feature importance scores from the Random Forest model.
    - Identify which features contribute most to prediction.
3. **Hyperparameter Tuning**
    - Experiment with `n_estimators`, `max_depth`, `min_samples_split`, and `min_samples_leaf`.
    - Observe effects on model performance and overfitting.
4. **Random Forest Regression**
    - Apply `RandomForestRegressor` on a regression dataset (e.g., Boston Housing).
    - Predict target values and evaluate using MSE, RMSE, and $R^2$ score.
5. **Comparing with Single Decision Tree**
    - Compare the performance of a single decision tree vs. random forest.
    - Visualize improvements in accuracy and reduction of overfitting.

## Lab – 08: Support Vector Machine (SVM)

**Aim:** To implement and evaluate SVM models for classification tasks using Python.

### Experiments

1. **Binary Classification using SVM**
    - Load a binary classification dataset (e.g., `Iris` for two classes).
    - Fit a `SVC` model using `sklearn.svm.SVC` with a linear kernel.
    - Predict and evaluate accuracy.
2. **Using Different Kernels**
    - Apply linear, polynomial, and RBF kernels.
    - Compare performance metrics for each kernel on the same dataset.
3. **Tuning Hyperparameters**
    - Experiment with `C`, `gamma`, and `degree` parameters.
    - Observe their effect on decision boundaries and model accuracy.
4. **Multi-class Classification with SVM**
    - Apply SVM on a multi-class dataset (e.g., full Iris dataset).
    - Use one-vs-one (OvO) or one-vs-rest (OvR) strategy for classification.
5. **Visualizing Decision Boundaries**

- Plot the decision boundary for a 2D feature dataset.
- Show support vectors and how they separate classes.

## Lab – 09: K-Nearest Neighbors (KNN)

**Aim:** To implement and evaluate KNN models for classification and regression tasks using Python.

### Experiments

1. **Implementing KNN for Classification**
   - Load a dataset (e.g., `Iris`).
   - Fit a `KNeighborsClassifier` using `sklearn.neighbors.KNeighborsClassifier`.
   - Predict class labels and evaluate accuracy.
2. **Choosing Different K Values**
   - Experiment with different values of `K` (e.g., 3, 5, 7).
   - Observe the effect of `K` on model performance and overfitting/underfitting.
3. **Distance Metrics in KNN**
   - Use different distance metrics (`euclidean`, `manhattan`, `minkowski`).
   - Compare their impact on classification accuracy.
4. **KNN for Regression**
   - Apply `KNeighborsRegressor` on a regression dataset (e.g., Boston Housing).
   - Predict target values and evaluate using MSE, RMSE, and $R^2$ score.
5. **Visualizing KNN Decision Boundaries**
   - Plot decision boundaries for a 2D dataset.
   - Show how KNN classifies regions based on nearest neighbors.

## Lab – 10: Unsupervised Learning – Clustering

**Aim:** To implement and evaluate clustering algorithms for grouping unlabeled data.

### Experiments

1. **K-Means Clustering**
   - Load a dataset (e.g., `Iris` without labels).
   - Apply `KMeans` from `sklearn.cluster` to group data into clusters.
   - Visualize clusters using scatter plots.
2. **Choosing Optimal Number of Clusters (Elbow Method)**
   - Apply the elbow method to determine the best `K` value.
   - Plot within-cluster sum of squares (WCSS) vs. number of clusters.
3. **Hierarchical Clustering**
   - Apply hierarchical clustering using `scipy.cluster.hierarchy`.
   - Plot dendrograms to visualize cluster formation.
4. **DBSCAN Clustering**
   - Apply `DBSCAN` algorithm for density-based clustering.

- Observe detection of noise points and clusters of varying densities.

5. **Evaluating Clustering Performance**
   - Use metrics like Silhouette Score, Davies-Bouldin Index to evaluate clustering quality.
   - Compare performance of K-Means, Hierarchical, and DBSCAN on the same dataset.

## Lab – 11: Dimensionality Reduction

**Aim:** To implement and evaluate dimensionality reduction techniques to reduce feature space while retaining important information.

**Experiments**

1. **Principal Component Analysis (PCA)**
   - Load a high-dimensional dataset.
   - Apply PCA using `sklearn.decomposition.PCA`.
   - Reduce dimensions and visualize the first two principal components.
2. **Variance Explained by Principal Components**
   - Calculate and plot the cumulative variance explained by each principal component.
   - Determine the number of components required to retain 90–95% of variance.
3. **Dimensionality Reduction for Classification**
   - Apply PCA on a labeled dataset (e.g., `Iris`).
   - Train a classifier (e.g., Logistic Regression) on reduced features and evaluate performance.
4. **t-Distributed Stochastic Neighbor Embedding (t-SNE)**
   - Apply t-SNE using `sklearn.manifold.TSNE` to visualize high-dimensional data in 2D.
   - Compare cluster separation with PCA visualization.
5. **Comparison of PCA and t-SNE**
   - Compare the results of PCA and t-SNE in terms of preserving structure and separability of classes.
   - Discuss suitability of each method for visualization vs preprocessing.

## Lab – 12: Model Evaluation and Cross-Validation

**Aim:** To evaluate machine learning models using various metrics and validate model performance using cross-validation.

**Experiments**

1. **Train-Test Split Evaluation**
   - Split a dataset into training and testing sets using `train_test_split`.
   - Train a model (e.g., Logistic Regression or Decision Tree) and evaluate accuracy, precision, recall, and F1-score on the test set.

2. **K-Fold Cross-Validation**
   - Apply `KFold` cross-validation using `sklearn.model_selection.KFold`.
   - Evaluate model performance across folds and compute average accuracy.
3. **Stratified K-Fold Cross-Validation**
   - Apply `StratifiedKFold` for classification datasets to maintain class distribution.
   - Compare results with regular K-Fold.
4. **Hyperparameter Tuning with Grid Search**
   - Use `GridSearchCV` to find optimal hyperparameters for a model (e.g., SVM, Random Forest).
   - Evaluate model performance with best parameters.
5. **ROC Curve and AUC for Model Evaluation**
   - Plot ROC curves for classifier models.
   - Calculate and interpret AUC scores to assess model discrimination ability.