***Project Structure and Flow***

***Step-by-Step Documentation***

**Step 1**: **Import Libraries**

The first step involves importing the necessary libraries required for data processing, visualization, model training, and evaluation.

**Sub-Steps**:

**1.1** Import libraries for handling data, visualizing, and splitting datasets:

- pandas: To work with dataframes.

- seaborn and matplotlib.pyplot: For visualizations.

- sklearn.datasets: To load the Iris dataset.

- sklearn.model\_selection: For splitting the data into training and test sets.

**1.2** Import libraries for preprocessing and evaluation:

- StandardScaler: To standardize features by removing the mean and scaling to unit variance.

- LabelEncoder: To convert categorical labels into numeric format.

**1.3** Import libraries for evaluation metrics:

- accuracy\_score, classification\_report, confusion\_matrix, ConfusionMatrixDisplay: To evaluate model performance.

**Step 2: Load and Preprocess Data**

We load the famous Iris dataset, preprocess it by converting the target into a more readable format, and explore the data.

**Sub-Steps:**

**2.1** Load the Iris dataset using load\_iris().

**2.2** Convert the dataset into a pandas DataFrame and rename the target labels for clarity (i.e., setosa, versicolor, and virginica).

**2.3** Display the first few rows of the dataset using df.head() to confirm the structure.

**2.4** Perform basic statistical analysis using df.describe() to understand the data distribution.

**2.5** Visualize the data distribution using Seaborn’s pairplot() to view relationships between features based on species.

**2.6** Check for missing values in the dataset using df.isnull().sum() to ensure data quality.

**Step 3: Split and Scale the Data**

We split the data into features (X) and labels (y), then divide it into training and test sets.

**Sub-Steps:**

**3.1** Define the feature set X by dropping the target column species, and define the label set y as the target column.

**3.2** Split the dataset into training and testing sets (80% training, 20% testing) using train\_test\_split().

**3.3** Standardize the features using StandardScaler() to ensure all features are on the same scale.

**Step 4: Model Training and Evaluation**

We train different machine learning models on the training data and evaluate their performance on the test data.

**Step 4.1: Random Forest Classifier**

**4.1.1** Initialize and train a RandomForestClassifier() with default hyperparameters.

**4.1.2** Predict labels on the test set and calculate accuracy using accuracy\_score().

**4.1.3** Generate and display a classification report using classification\_report() to analyze model performance for each class.

**Step 4.2: K-Nearest Neighbors (KNN)**

**4.2.1** Initialize and train a KNeighborsClassifier() with n\_neighbors=5.

**4.2.2** Predict the labels on the test set and calculate accuracy using accuracy\_score().

**4.2.3** Display the classification report to evaluate KNN’s performance.

**Step 4.3: Support Vector Machine (SVM)**

**4.3.1** Train an SVC() with a linear kernel on the training data.

**4.3.2** Predict labels on the test set and calculate the accuracy.

**4.3.3** Generate and analyze the classification report.

**Step 5: Hyperparameter Tuning (Random Forest)**

We perform hyperparameter tuning using GridSearchCV to find the optimal set of parameters for the Random Forest model.

**Sub-Steps:**

**5.1** Define a parameter grid (param\_grid) containing different values for n\_estimators, max\_depth, min\_samples\_split, and min\_samples\_leaf.

**5.2** Use GridSearchCV() to search for the best hyperparameters with cross-validation (5-fold).

**5.3** Train the model on the training set and extract the best parameters using grid\_search.best\_params\_.

**5.4** Evaluate the tuned Random Forest model on the test set and display the accuracy and classification report.

**Step 6: Neural Network Implementation**

We build and train a simple neural network using TensorFlow’s Keras API.

**Sub-Steps:**

**6.1** Encode the target labels (species) using LabelEncoder() and convert them into a categorical format using to\_categorical().

**6.2** Define the neural network architecture using Sequential() with three layers:

- Input Layer: 32 units, ReLU activation.

- Hidden Layer: 16 units, ReLU activation.

- Output Layer: 3 units (since there are 3 classes), softmax activation.

**6.3** Compile the model using adam optimizer and categorical\_crossentropy loss function.

**6.4** Train the neural network for 100 epochs with a batch size of 10, and display the training progress.

**6.5** Evaluate the neural network on the test data and calculate accuracy.

**Step 7: Confusion Matrix Visualization**

We visualize the confusion matrices for each model (Random Forest, KNN, SVM, and Neural Network).

**Sub-Steps:**

**7.1** Generate the confusion matrix for the Random Forest model using confusion\_matrix(), and display it using ConfusionMatrixDisplay().

**7.2** Repeat the process for KNN, SVM, and the Neural Network models. For the neural network, convert the predicted probabilities into class labels using argmax() before generating the confusion matrix.

**Step 8: Cross-Validation**

We perform cross-validation to evaluate model performance stability across different splits of the data.

**Sub-Steps:**

**8.1** Perform 5-fold cross-validation for the Random Forest model using cross\_val\_score() and display the mean cross-validation score.

**8.2** Repeat the same process for KNN and SVM models.

**Step 9: Feature Importance**

We analyze the feature importance from the trained Random Forest model.

**Sub-Steps:**

**9.1** Extract the feature importances from the RandomForestClassifier() using feature\_importances\_.

**9.2** Sort and plot the feature importances using matplotlib.

**Step 10: Visualization of Decision Boundaries (SVM)**

We visualize decision boundaries for the SVM model in a 2D space using a mesh grid.

**Sub-Steps:**

**10.1** Generate a mesh grid using np.meshgrid() for plotting the decision boundaries.

**10.2** Predict the class labels on the grid points using the trained SVM model.

**10.3** Plot the decision boundaries using contourf() and color-code the regions based on the predicted class labels.

**Step 11: Model Comparison**

We compare the performance of all models based on accuracy.

**Sub-Steps:**

**11.1** Display the accuracy of each model (Random Forest, KNN, SVM, and Neural Network) to see which model performs best on the test data.

**Step 12: Conclusion**

Finally, we conclude by reviewing the performance of each model, the impact of hyperparameter tuning, and model comparison.