**A Comparative Study of Six ML Models for Flower Classification**

|  |
| --- |
| **ABSTRACT**   * **Project Scope:** The project scope entails employing diverse machine learning algorithms on the Iris dataset to classify iris flower species. With 200 instances categorized into Setosa, Versicolor, and Virginica classes, the study includes model training, evaluation, and comparative analysis to discern the most effective algorithms, offering insights for botany and related fields. The scope encompasses the following key elements:   + **Dataset Analysis**: Utilizing the Iris dataset containing 200 instances of iris flowers, each categorized into one of three classes: Setosa, Versicolor, and Virginica. The dataset includes features such as petal and sepal measurements   + **Model Training and Evaluation:** Segmentation of the dataset into training and testing subsets, followed by the training of machine learning models on the training data. The performance of each model is evaluated using metrics like accuracy, precision, and recall on the testing data.   + **Comparative Analysis:** Conducting a comparative analysis of the performance of different machine learning algorithms in iris flower classification. This involves identifying the strengths and weaknesses of each algorithm in accurately categorizing iris flower species.. * **Objectives:** This project has several well-defined objectives:   + **Primary Objective:** Implement various machine learning algorithms for iris flower classification.   + **Learning and Exploration:**     - Gain insights into the effectiveness of logistic regression, k-nearest neighbors, random forest, SVM, naive Bayes, and decision tree algorithms in iris flower classification.     - Explore the nuances of each algorithm and their suitability for the task.   + **Performance Metrics:**     - Evaluate the performance of the implemented algorithms using metrics such as accuracy, precision, and recall.     - Determine which algorithms yield the highest classification accuracy for iris flower species.   + **Iterative Development:** Engage in iterative development by refining the implementation based on performance metrics and comparative analysis. Continuously improve the classification models to achieve optimal accuracy and reliability. * **Methods:** The project aims to systematically evaluate and compare the effectiveness of different machine learning techniques for iris flower classification. Key methodologies include:   + **Data Preprocessing:**     - Perform data cleaning to handle missing values and outliers.     - Standardize or normalize the feature values to ensure consistency across the dataset.     - Split the dataset into training and testing subsets for model evaluation.   + **Algorithm Selection:**     - Choose a variety of machine learning algorithms suitable for classification tasks, including logistic regression, k-nearest neighbors, random forest, SVM, naive Bayes, and decision tree algorithms.     - Consider the characteristics of the dataset and the nature of the classification problem when selecting algorithms.   + **Model Training and Evaluation:**     - Train each selected algorithm using the training subset of the dataset.     - Adjust hyperparameters to optimize model performance, utilizing techniques such as cross-validation to prevent overfitting.     - Evaluate the trained models using performance metrics such as accuracy, precision, recall, and F1-score.     - Utilize confusion matrices and ROC curves to assess the models' classification performance comprehensively.   **I INTRODUCTION**   * Machine learning (ML) has emerged as a transformative force, revolutionizing various domains with its ability to extract patterns and insights from data. In this project, we embark on a journey through the landscape of ML techniques, tracing their historical evolution and leveraging their power to address the fundamental challenge of iris flower classification. Rooted in the rich history of ML, our endeavor integrates advanced algorithms and Python programming, symbolizing the convergence of tradition and innovation in the pursuit of scientific inquiry. * The roots of machine learning can be traced back to the mid-20th century, where pioneers such as Alan Turing and Arthur Samuel laid the groundwork for computational systems capable of learning from data. The advent of neural networks in the 1980s marked a significant milestone, followed by the resurgence of interest in ML driven by advancements in computing power and data availability in the 21st century. Today, ML stands at the forefront of technological innovation, permeating diverse fields ranging from healthcare to finance, and beyond. * The classification of iris flower species represents a quintessential challenge in ML, epitomizing the paradigm of supervised learning. Our task entails discerning between three distinct iris species—Setosa, Versicolor, and Virginica—based on their morphological attributes. This endeavor is characterized by its complexity stemming from the subtle variations in petal and sepal measurements across different species, underscoring the need for sophisticated ML techniques to achieve accurate classification. * Python has emerged as the de facto programming language for ML, owing to its simplicity, versatility, and robust ecosystem of libraries such as scikit-learn and TensorFlow. Guido van Rossum's creation has witnessed widespread adoption within the ML community, facilitating seamless integration of algorithms and enabling rapid prototyping and experimentation. Python's ascendancy in ML reflects its capacity to democratize access to cutting-edge technologies and empower researchers and practitioners to unlock the full potential of machine learning. * Through the synergistic fusion of advanced algorithms, Python programming, and a deep appreciation for the complexities of nature, we endeavor to unravel the mysteries of floral taxonomy and contribute to the collective body of knowledge in both ML and botanical sciences.   **III PROBLEM STATEMENT**   * **The Problem:**   The task at hand revolves around the intricate challenge of accurately classifying iris flower species based on their morphological attributes. Specifically, the objective is to develop a robust machine learning model capable of distinguishing between three distinct iris species—Setosa, Versicolor, and Virginica—using features such as petal length, petal width, sepal length, and sepal width. The classification of iris species serves as a quintessential problem in supervised learning, necessitating the exploration and implementation of sophisticated machine learning techniques.  The inherent complexity of the problem stems from the subtle variations in petal and sepal measurements across different iris species, posing a formidable challenge for traditional classification methods. Moreover, the dataset comprising 200 meticulously annotated instances further amplifies the intricacy of the task, requiring a nuanced approach to feature extraction, model training, and evaluation.  The successful resolution of this problem holds significant implications for both the field of machine learning and botanical sciences. Accurate classification of iris species not only showcases the efficacy of advanced ML algorithms but also contributes to our understanding of floral diversity and taxonomy. Furthermore, the development of a reliable classification model lays the groundwork for broader applications in species identification, biodiversity conservation, and ecological research.  Thus, the problem statement encapsulates a multifaceted challenge with far-reaching implications, underscoring the importance of leveraging cutting-edge technologies to address real-world problems in interdisciplinary domains..   * **The Project's Goal:** The project seeks to not only address the immediate challenge of iris flower classification but also pave the way for future advancements in the intersection of machine learning and botanical sciences. Specifically, the goals include:   + **Automation:** Develop a machine learning system capable of automating the classification of iris flower species, reducing the need for manual intervention in the identification process.   + **Accuracy:** Achieve high levels of accuracy in iris species classification, ensuring reliable and consistent results across different instances and datasets.   + **Efficiency:** Optimize the performance of the machine learning algorithms to ensure computational efficiency, minimizing processing time and resource utilization.   + **Scalability:** Design the classification system to be scalable, capable of handling larger datasets and accommodating future growth in data volume and complexity.   + **Foundation for Further Applications:** Establish a solid foundation for further applications of machine learning in botanical sciences and related domains. Provide insights and methodologies that can be extended to other classification tasks, contributing to broader advancements in artificial intelligence research.     **IV PROPOSED SOLUTION**   * **Overview:** The proposed solution utilizes a combination of machine learning algorithms to classify iris flower species based on their morphological attributes. Here's a breakdown of how these algorithms achieve accurate classification::   + **Algorithm Selection:**     - Logistic Regression, k-Nearest Neighbors, Random Forest, Support Vector Machine, Naive Bayes, and Decision Tree algorithms are selected for their suitability in classification tasks.     - Each algorithm brings unique strengths to the task, from the simplicity of logistic regression to the complexity of ensemble methods like random forest.   + **How Training Works:**     1. **Initialization:** The training process begins with the initialization of the machine learning algorithms' parameters. For instance, in logistic regression, the weights are initialized randomly.     2. **Forward Pass:** Labeled instances from the training dataset are fed into each algorithm. The algorithms process the input data through their respective mathematical computations, progressively uncovering patterns in the relationships between petal and sepal measurements and iris species.     3. **Prediction:** After processing the input data, each algorithm produces a prediction or probability distribution over the possible iris flower species. For example, a logistic regression model might output probabilities for each class, indicating the likelihood of an instance belonging to Setosa, Versicolor, or Virginica.     4. **Error Calculation:** The predictions made by the algorithms are compared to the true labels of the training instances. The error or loss is calculated using a suitable loss function, such as cross-entropy loss for multi-class classification tasks. This error quantifies the discrepancy between the predicted and true labels for each training instance.     5. **Backpropagation:** Backpropagation is a key algorithmic technique used to update the parameters (e.g., weights) of the algorithms based on the calculated error. It involves computing the gradients of the loss function with respect to the model parameters. These gradients provide information on how to adjust the parameters to minimize the error.     6. **Repeat:** The training process iterates over the entire training dataset multiple times, known as epochs. With each epoch, the algorithms update their parameters based on the gradients computed from different batches of training data. * **Key Components:** The process of building and utilizing the iris classification solution involves several crucial steps:   1. **Data Collection and Preprocessing:**      + Collecting a comprehensive dataset of iris flower instances, including petal and sepal measurements for each sample.      + Preprocessing the dataset by handling missing values, removing outliers, and standardizing or normalizing the feature values to ensure consistency and facilitate algorithm convergence.   2. **Algorithm Selection:**      + Choosing suitable machine learning algorithms for iris flower classification, such as logistic regression, k-nearest neighbors, random forest, support vector machine, naive Bayes, and decision tree algorithms.      + Each algorithm is selected based on its appropriateness for the classification task and its potential to achieve high accuracy.   3. **Model Training:**      + Dividing the dataset into training and testing subsets to facilitate model training and evaluation.      + Training each selected algorithm using the training data, allowing the models to learn from the patterns present in the input features and their corresponding iris flower species labels.   4. **Evaluation Metrics:**      + Employing appropriate evaluation metrics, such as accuracy, precision, recall, and F1-score, to assess the performance of the trained models.      + These metrics provide insights into the algorithms' ability to correctly classify iris flower species and their overall effectiveness in solving the classification task.   **V PROPOSED ALGORITHM**  Segmentation serves to eliminate the unwanted background, isolating the focal point (foreground) object, namely the flower. Its primary aim is to streamline the depiction of the flower, presenting a more discernible and analytically manageable entity.  In the process of Feature Extraction, we derive essential characteristics or information from the flower, represented as real values such as floats, integers, or binaries. Key features used for quantifying plants or flowers include color, shape, and texture. Rather than relying solely on a single feature vector, we opt for a comprehensive approach, combining different feature descriptors to more effectively identify the image. Table 1 showcases the representation of the initial five Iris datasets.Table 1  Upon extracting features and labels from the Iris dataset, the subsequent step entails training the system. Utilizing scikit-learn, we construct machine learning models capable of classifying Iris flowers into their respective subspecies. The descriptive statistics of the Iris dataset are delineated in Table 2 below.  Feature extraction methodically extracts pertinent properties or data points from the flowers, encapsulating them as real values such as floats, integers, or binaries. Employing a diverse array of techniques including Logistic Regression, Decision Tree, k-Nearest Neighbor, Random Forest Classifier, Gaussian Naive Bayes, and Linear SVC further bolsters the accuracy and reliability of the classification process.   1. **Decision tree**   The Decision Tree stands out as the primary tool for AI and ML-based predictions, renowned for its effectiveness and widespread usage. Each leaf node holds a class label, while branches denote outcomes, and internal nodes represent attribute tests. This tree-based structure resembles flow charts and finds applications in both regression and classification tasks. Notably, decision trees are characterized by their simplicity and lack of parameters, enabling the creation of models that forecast  variable values using straightforward decision rules.   1. **Random forest classifier**   In the training phase, the ensemble learning approach of random forest constructs decision tree models. These models collectively contribute to the final decision-making process. Random forest, a classification method in machine learning, operates by creating an ensemble of trees. Unlike decision trees that utilize the entire dataset and consider all attributes, random forest uses only a subset of the data and a limited number of features.   1. **Gaussian Naive Bayes**   Gaussian Naive Bayes, a statistics-driven classification method rooted in Bayes Theorem, operates under stringent independence assumptions. These conditions dictate that alterations in one value do not affect another. While the performance of naive Bayes classifiers may decline with larger training sets, they remain renowned for their expressive nature, scalability, and moderate accuracy in machine learning. The efficiency of these classifiers is influenced by various factors.   1. **Logistic Regression**   Logistic regression estimates event occurrence likelihood using a logistic function. It employs predictor variables, whether numerical or categorical, akin to other regression types. Specifically designed for binary data, it determines event occurrence (1) or non-occurrence (0) based on features.   1. **Iteration and Improvement**   K-Nearest Neighbors (KNN) is a supervised learning technique widely employed for classification tasks, utilizing the entire dataset during prediction. When presented with new data, KNN searches for the k most similar instances within the training dataset and predicts the class of the new data based on the majority class among its nearest neighbors. This method retains all available examples and determines the classification of new instances based on their similarity to existing data points.   1. **Support Vector Machine**   Support Vector Machine (SVM) is a supervised learning method commonly utilized in classification tasks, which operates by analyzing the entire dataset during the training phase. When making predictions on unseen data, SVM identifies the optimal hyperplane that best separates different classes in the feature space. It then assigns the class label to the new data point based on its position relative to this hyperplane. SVM retains crucial support vectors from the training data to define the decision boundary for classifying new instances.  **VIII SOURCE CODE**  """ Step 1: Import Library"""  import numpy as np  import pandas as pd  import matplotlib.pylab as plt  import seaborn as sns  """# \*\*Step 2 : Read Dataset\*\*"""  data = pd.read\_csv("/content/IRIS Flower Dataset.csv")  df = pd.DataFrame(data)  df["species"].replace({"Iris-setosa":0 , "Iris-versicolor":1 , "Iris-virginica":2} , inplace = True)  df  from google.colab import drive  drive.mount('/content/drive')  """# \*\*Step 3 : Dataset Overview\*\*"""  df.describe(include = 'all')  df.hist()  plt.show()  plt.figure (figsize = (16 , 6))  mask = np.triu(np.ones\_like(df.corr(method = "spearman") , dtype = bool))  heatmap = sns.heatmap(df.corr(method = "spearman") , mask = mask , vmin = -1 , vmax = 1 ,  annot = True , cmap="BrBG")  heatmap.set\_title("Triangle Correlation Heatmap" , fontdict = {'fontsize': 18} , pad =16 )  sns.pairplot(df , hue='species' , diag\_kind="hist" , corner=True , palette = 'hls')  #hist - Histogram  #hls - Hue, lightroom, Saturation  sns.pairplot(df , hue='species' , diag\_kind="kde" , corner=True , palette = ['blue','green','red'])  #kde means Kernal desnity estimation plot  """# \*\*Step 4 : Data science & Visualization\*\*"""  Num = ['sepal\_length' , 'sepal\_width' , 'petal\_length' , 'petal\_width']  j = 0  while j < 5:  fig = plt.figure (figsize = [10 , 4])  plt.subplot(1, 2, 1)  sns.boxplot (x = Num[j] , data = df , color='skyblue')  sns.set(font\_scale=1.25)  j += 1  plt.subplot(1, 2, 2)  sns.boxplot (x = Num[j] , data = df , color='skyblue')  sns.set(font\_scale=1.25)  j += 1  if j == 4:  break  plt.show()  sns.countplot(x = df['species'] , data = df)  """# \*\*Step 5 : Models\*\*"""  X = pd.DataFrame(df , columns = ["sepal\_length" , "sepal\_width" , "petal\_length" , "petal\_width"])  y = df["species"].values.reshape(-1,1)  Scaler = preprocessing.MinMaxScaler(feature\_range = (0,1))  Norm1 = Scaler.fit\_transform(df)  Norm1\_df=pd.DataFrame (Norm1 , columns = ["species" , "sepal\_length" , "sepal\_width" , "petal\_length" , "petal\_width"])  Norm1\_df.head()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y , test\_size=0.5 , random\_state = 0)  def Evaluate\_Performance(Model, Xtrain, Xtest, Ytrain, Ytest) :  Model.fit(Xtrain,Ytrain)  overall\_score = cross\_val\_score(Model, Xtrain,Ytrain, cv=10)  model\_score = np.average(overall\_score)  Ypredicted = Model.predict(Xtest)  avg = 'weighted'  print("\n • Training Accuracy Score : ", round(Model.score(Xtrain, Ytrain) \* 100,2))  print(f" • Cross Validation Score : {round(model\_score \* 100,2)}")  print(f" • Testing Accuracy Score :{round(accuracy\_score(Ytest, Ypredicted) \* 100,2)}")  print(f" • Precision Score is : {np.round(precision\_score(Ytest, Ypredicted , average=avg) \* 100,2)}")  print(f" • Recall Score is : {np.round(recall\_score(Ytest, Ypredicted , average=avg) \* 100,2)}")  print(f" • F1-Score Score is : {np.round(f1\_score(Ytest, Ypredicted , average=avg) \* 100,2)}")  """# \*\*Logestic Regression\*\*"""  LogReg = LogisticRegression(solver = "liblinear" , C=50)  LogReg.fit(X\_train , y\_train.ravel())  y\_pred\_LR = LogReg.predict(X\_test)  print("Logistic Regression : ")  Evaluate\_Performance(LogReg, X\_train, X\_test, y\_train, y\_test)  kfold = KFold(37)  LR\_r = cross\_val\_score (LogReg, X, y, cv = kfold)  print(np.std(LR\_r))  cm = confusion\_matrix (y , LogReg.predict(X))  fig, ax = plt.subplots (figsize = (8, 8))  ax.imshow(cm)  ax.grid(False)  ax.set\_xlabel('Predicted outputs', fontsize= 14 , color='black')  ax.set\_ylabel('Actual outputs', fontsize= 14 , color='black')  ax.xaxis.set(ticks=range(3))  ax.yaxis.set(ticks=range(3))  ax.set\_ylim(2.5 , -0.5)  for i in range(3):  for j in range(3):  ax.text(j, i, cm[i, j], ha = 'center' , va = 'center' , color = 'red')  plt.show()  """# \*\*K Nearest Neighbors\*\*"""  training\_acc = []  test\_acc = []  neighbors\_setting = range(1,30)  for n\_neighbors in neighbors\_setting:  KNN = KNeighborsClassifier(n\_neighbors = n\_neighbors)  KNN.fit(X\_train , y\_train.ravel())  training\_acc.append(KNN.score(X\_train , y\_train))  test\_acc.append(KNN.score(X\_test , y\_test))  plt.plot(neighbors\_setting , training\_acc , label = "Accuracy of the training set")  plt.plot(neighbors\_setting , test\_acc , label = "Accuracy of the test set")  plt.xlabel("Number of neighbors")  plt.ylabel("Accuracy")  plt.grid(linestyle=':')  plt.legend()  parameters = {"n\_neighbors" : range(1,50)}  grid\_kn = GridSearchCV(estimator = KNN , param\_grid = parameters , scoring = "accuracy" , cv = 5 , verbose = 1 , n\_jobs = -1)  grid\_kn.fit(X\_train , y\_train.ravel())  grid\_kn.best\_params\_  K = 3  KNN = KNeighborsClassifier(K)  KNN.fit(X\_train , y\_train.ravel())  y\_pred\_KNN = KNN.predict(X\_test)  print("K-Nearest Neighbors : ")  Evaluate\_Performance(KNN, X\_train, X\_test, y\_train, y\_test)  import matplotlib.pyplot as plt  import numpy as np  unique, counts = np.unique(y\_pred\_KNN, return\_counts=True)  plt.bar(unique, counts, color='blue')  plt.xlabel('Predicted Classes')  plt.ylabel('Count')  plt.title('Distribution of KNN Classes')  for i, count in enumerate(counts):  plt.text(unique[i], count, str(count), ha='center', va='bottom')  plt.show()  KNN\_r = cross\_val\_score (KNN, X, y, cv = 10)  K = np.std(KNN\_r)  print(K)  cm = confusion\_matrix (y , KNN.predict(X))  fig, ax = plt.subplots (figsize = (8, 8))  ax.imshow(cm)  ax.grid(False)  ax.set\_xlabel('Predicted outputs', fontsize= 14 , color='black')  ax.set\_ylabel('Actual outputs', fontsize= 14 , color='black')  ax.xaxis.set(ticks=range(3))  ax.yaxis.set(ticks=range(3))  ax.set\_ylim(2.5 , -0.5)  for i in range(3):  for j in range(3):  ax.text(j, i, cm[i, j], ha = 'center' , va = 'center' , color = 'red')  plt.show()  """# \*\*Naive Bayes\*\*"""  NB = GaussianNB()  NB.fit(X\_train , y\_train.ravel())  y\_pred\_NB = NB.predict(X\_test)  print("Naive Bayes : ")  Evaluate\_Performance(NB, X\_train, X\_test, y\_train, y\_test)  NB\_r = cross\_val\_score (NB, X, y, cv = 10)  N = np.std(NB\_r)  print(N)  cm = confusion\_matrix (y , NB.predict(X))  fig, ax = plt.subplots (figsize = (8, 8))  ax.imshow(cm)  ax.grid(False)  ax.set\_xlabel('Predicted outputs', fontsize= 14 , color='black')  ax.set\_ylabel('Actual outputs', fontsize= 14 , color='black')  ax.xaxis.set(ticks=range(3))  ax.yaxis.set(ticks=range(3))  ax.set\_ylim(2.5 , -0.5)  for i in range(3):  for j in range(3):  ax.text(j, i, cm[i, j], ha = 'center' , va = 'center' , color = 'red')  plt.show()  """# \*\*Support Vector Machine\*\*"""  SVM = SVC()  SVM.fit(X\_train , y\_train)  y\_pred\_SVM = SVM.predict(X\_test)  print("SVM : ")  Evaluate\_Performance(SVM, X\_train, X\_test, y\_train, y\_test)  SVM\_r = cross\_val\_score (SVM, X, y, cv = 10)  S = np.std(SVM\_r)  print(S)  cm = confusion\_matrix (y , SVM.predict(X))  fig, ax = plt.subplots (figsize = (8, 8))  ax.imshow(cm)  ax.grid(False)  ax.set\_xlabel('Predicted outputs', fontsize= 14 , color='black')  ax.set\_ylabel('Actual outputs', fontsize= 14 , color='black')  ax.xaxis.set(ticks=range(3))  ax.yaxis.set(ticks=range(3))  ax.set\_ylim(2.5 , -0.5)  for i in range(3):  for j in range(3):  ax.text(j, i, cm[i, j], ha = 'center' , va = 'center' , color = 'red')  plt.show()  """# \*\*Decision Tree\*\*"""  DT = DecisionTreeClassifier(max\_depth = 3)  DT = DT.fit(X\_train , y\_train)  y\_pred\_DT = DT.predict(X\_test)  print("Decision Tree : ")  Evaluate\_Performance(DT, X\_train, X\_test, y\_train, y\_test)  DT\_r = cross\_val\_score (DT, X, y, cv = 10)  D = np.std(DT\_r)  print(D)  cm = confusion\_matrix (y , DT.predict(X))  fig, ax = plt.subplots (figsize = (6, 6))  ax.imshow(cm)  ax.grid(False)  ax.set\_xlabel('Predicted outputs', fontsize= 14 , color='black')  ax.set\_ylabel('Actual outputs', fontsize= 14 , color='black')  ax.xaxis.set(ticks=range(3))  ax.yaxis.set(ticks=range(3))  ax.set\_ylim(2.5 , -0.5)  for i in range(3):  for j in range(3):  ax.text(j, i, cm[i, j], ha = 'center' , va = 'center' , color = 'red')  plt.show()  F = ["sepal\_length" , "sepal\_width" , "petal\_length" , "petal\_width"]  T = ['0' , '1' , '2']  fig = plt.figure(figsize = (10 , 10))  plot = tree.plot\_tree (DT , feature\_names = F , class\_names = T , filled = True)  """# \*\*Random Forest\*\*"""  RF = RandomForestClassifier(n\_estimators = 400, max\_depth = 3)  RF = RF.fit(X\_train , y\_train)  y\_pred\_RF = RF.predict(X\_test)  print("Random Forest : ")  Evaluate\_Performance(RF, X\_train, X\_test, y\_train, y\_test)  RF\_r = cross\_val\_score (RF, X, y, cv = 10)  R = np.std(RF\_r)  print(R  cm = confusion\_matrix (y , RF.predict(X))  fig, ax = plt.subplots (figsize = (6, 6))  ax.imshow(cm)  ax.grid(False)  ax.set\_xlabel('Predicted outputs', fontsize= 14 , color='black')  ax.set\_ylabel('Actual outputs', fontsize= 14 , color='black')  ax.xaxis.set(ticks=range(3))  ax.yaxis.set(ticks=range(3))  ax.set\_ylim(2.5 , -0.5)  for i in range(3):  for j in range(3):  ax.text(j, i, cm[i, j], ha = 'center' , va = 'center' , color = 'red')  plt.show()  #white box one  F = ["sepal\_length" , "sepal\_width" , "petal\_length" , "petal\_width"]  T = ['0' , '1' , '2']  fig = plt.figure(figsize = (10 , 10))  plot = tree.plot\_tree (RF.estimators\_[5] , feature\_names = F , class\_names = T , filled = True)  """# \*\*Step 6 : Conclusion\*\*"""  models = pd.DataFrame ({'Model' : ['Logestic Regression' , ' KNN' , 'Naive Bayes' , 'SVM' , 'Decision Tree' , 'Random Forest'] ,  'Precision' : [precision\_score(y\_test, y\_pred\_LR, average='weighted') , precision\_score(y\_test, y\_pred\_KNN, average='weighted') , precision\_score(y\_test, y\_pred\_NB, average='weighted') , precision\_score(y\_test, y\_pred\_SVM, average='weighted') , precision\_score(y\_test, y\_pred\_DT, average='weighted') , precision\_score(y\_test, y\_pred\_RF, average='weighted')] ,  'Recall' : [recall\_score(y\_test, y\_pred\_LR, average='weighted') , recall\_score(y\_test, y\_pred\_KNN, average='weighted') , recall\_score(y\_test, y\_pred\_NB, average='weighted') , recall\_score(y\_test, y\_pred\_SVM, average='weighted') , recall\_score(y\_test, y\_pred\_DT, average='weighted') , recall\_score(y\_test, y\_pred\_RF, average='weighted')] ,  'F1-score' : [f1\_score(y\_test, y\_pred\_LR, average='weighted') , f1\_score(y\_test, y\_pred\_KNN, average='weighted') , f1\_score(y\_test, y\_pred\_NB, average='weighted') , f1\_score(y\_test, y\_pred\_SVM, average='weighted') , f1\_score(y\_test, y\_pred\_DT, average='weighted') , f1\_score(y\_test, y\_pred\_RF, average='weighted')] ,  'Accuracy' : [accuracy\_score(y\_test, y\_pred\_LR) , accuracy\_score(y\_test, y\_pred\_KNN) , accuracy\_score(y\_test, y\_pred\_NB) , accuracy\_score(y\_test, y\_pred\_SVM) , accuracy\_score(y\_test, y\_pred\_DT) , accuracy\_score(y\_test, y\_pred\_RF)],  'Err' : [np.std(LR\_r) , np.std(KNN\_r) , np.std(NB\_r) , np.std(SVM\_r) , np.std(DT\_r) , np.std(RF\_r)]})  print(tabulate(models, headers='keys', tablefmt='rst'))  fig, ax = plt.subplots(figsize=(10, 6), dpi= 80, facecolor='#99ccff')  ax.set\_facecolor('#66ccff')  ax.set\_title('Comparison of predicted models', fontdict={'size':22} , color='b')  ax.errorbar(models['Accuracy'], models['Model'], xerr = models['Err'] , fmt='o', marker='s', color='b'  , linewidth=2, capsize=3)  ax.set(xlim=(0.8, 1), xticks=np.arange(0.8, 1, step = 0.05))  plt.grid(color = '#333366', axis = 'y', linestyle = '--', linewidth = 0.5)  plt.show()  model\_names = ["Logestic Regression", "KNN", "Naive Bayes", "SVM", "Decision Tree", "Random Forest"]  accuracy\_values = [0.96, 0.93, 0.89, 0.94, 0.91, 0.95]  fig, axs = plt.subplots(2, 3, figsize=(15, 8))  for i, (model\_name, accuracy) in enumerate(zip(model\_names, accuracy\_values)):  ax = axs[i // 3, i % 3]  ax.pie([accuracy, 1 - accuracy], labels=["Accuracy", "Error"], autopct="%1.1f%%", startangle=90)  ax.set\_title(model\_name)  plt.tight\_layout()  plt.show()  **X RESULT**  **Graph:**  Accuracy    **XI CONCLUSION**   * In this project, we delved into the realm of supervised machine learning, specifically focusing on Iris Flower Classification. Leveraging six diverse machine learning models - k-Nearest Neighbors, Logistic Regression, Decision Tree, SVM, Naive Bayes, and Random Forest classifier - we aimed to construct robust models capable of accurately predicting iris flower species. Through extensive data visualization, analysis, and model construction, we gained valuable insights into the intricacies of the iris dataset and the performance of various algorithms. Our findings suggest that Logistic Regression emerges as the most accurate classifier among the six models tested. Moving forward, the overarching goal of supervised learning remains to build models that generalize well to unseen data, ensuring accurate predictions for future iris flowers beyond the scope of our training dataset.   **REFERENCES**  [1] Kaggle Dataset  [2] Redux: Kernels Edition – Dataset Description |
|  |
|  |