A picture containing text, logo, symbol, emblem

Description automatically generated

T.C.

TOROS UNIVERSITY

FACULTY OF ENGINEERING/ SOFTWARE ENGINEERING PROGRAM

STUDENTS PERFORMANCE IN EXAMS

İBRAHİM HALİL ALPA

205060007

FINAL PROJECT

JULY / 2023

DATASET PROPERTIES

I used the dataset named "Students' Performance in Exams" in my project. This dataset is a dataset that includes students' math score, reading score, and writing score.

This data set consists of 1000 students and exam results are recorded according to different characteristics of students such as gender, race/ethnicity, parental level of education, lunch, and test preparation course. These features can be used to analyze students' exam performance. For example, features such as students' gender and parents' education level can have an impact on test scores, and the data set can be used to determine these effects.

* "gender": refers to the student's gender (male or female).
* "race/ethnicity": The student's race/ethnicity (which may fall into five different categories (group A-group E)).
* "parental level of education" refers to the parents' educational background (high school, some college, etc.).
* "lunch": The student's lunch option (standard, free/reduced).
* "test preparation course": Whether the student finished (completed or none) a test preparation course.
* "math score": The student's performance on the math test
* "reading score": The student's performance on the reading test
* "writing score": The student's performance on the writing test

Finally, the data set I will use can be used for classification and regression analysis. Classification analysis aims to classify students into a particular class or category. Regression analysis can be used to predict students' test scores. I used classification analysis in this project.

A screenshot of a computer

Description automatically generated

HOW DID I SCALE THE ATTRIBUTES OF THE DATASET?

gender:

* female=1,
* male=2

race/ethnicity:

* group A=1,
* group B= 2
* group C=3,
* group D=4,
* group E=5

parental\_level\_of\_education:

* high school=1,
* some college=2,
* associate's degree=3,
* bachelor's degree=4,
* master's degree=5

lunch

* standard=1
* free/reduced=2

test preparation course

* none=1
* completed

USED LIBRARIES, MODULES, METHODS/FUNCTION, AND METRICS

LIBRARIES

Pandas Library

Pandas is a Python library that provides a powerful and user-friendly interface for data analysis and manipulation. It offers a high-performance data structure called DataFrame, which is a two-dimensional table-like structure for handling structured data. Pandas allows you to load, filter, transform, merge, and summarize datasets. It offers a wide range of functions for data manipulation, cleaning, preprocessing, and visualization. Pandas is widely used in data analysis tasks and integrates well with other libraries in the data science ecosystem. Overall, it is a valuable tool for working with structured data in Python.

Scikit-learn Library

Scikit-learn or Sklearn is a Python-based library used for building machine learning models. It provides a wide range of learning algorithms that can be applied to tasks such as regression, clustering, and classification. Sklearn is compatible with other popular Python libraries like NumPy and SciPy, allowing seamless integration with the broader data science ecosystem. With Sklearn, you can leverage a variety of machine learning techniques to analyze and process data, make predictions, and solve complex problems. It offers a comprehensive set of tools for model selection, evaluation, and optimization. Sklearn is a versatile and widely-used library that empowers users to implement various machine learning algorithms efficiently and effectively.

MODULES

sklearn.ensemble

sklearn.ensemble is a module in the scikit-learn library and is used to implement ensemble methods. Ensemble methods aim to combine multiple learners to create a more powerful and generalizing model. This module includes ensemble algorithms and can be used for classification, regression and other machine learning problems. Ensemble methods provide advantages such as better prediction performance, generalization ability and model strengthening. sklearn.ensemble module allows you to build more effective and performant machine learning models using ensemble methods.

sklearn.model\_selection

sklearn.model\_selection is a module of the scikit-learn library and is used for the selection, hyperparameter tuning and performance evaluation of machine learning models. This module provides functionality such as splitting the dataset into training and test subsets, cross-validation, hyperparameter search and calculation of performance metrics. In short, the sklearn.model\_selection module is a toolkit that facilitates important functions used for model selection and evaluation.

sklearn.metrics

sklearn.metrics is a module of the scikit-learn library and contains metrics used to evaluate the performance of machine learning models. It enables the calculation of various metrics for classification and regression. For example, there are classification metrics such as accuracy, precision, recall, F1 score and regression metrics such as mean absolute error, mean squared error, R-squared. sklearn.metrics module provides a useful set of tools for objectively evaluating model performance, comparing different models and model improvement.

sklearn.tree

sklearn.tree is a module of the scikit-learn library and is used to build machine learning models based on decision trees. Decision trees are tree-structured models that can classify or regress using the features of a dataset. sklearn.tree module contains the decision tree algorithm and classes. Thanks to this module, decision tree models can be created that can be used for classification and regression problems that make data-driven decisions

METHODS/FUNCTION

RandomForestClassifier

RandomForestClassifier is a classification method available in the scikit-learn library. Using the Random Forest algorithm, one of the ensemble methods, it creates a more powerful classification model by combining multiple decision trees. Trees trained with random sampling and different features classify by combining the predictions of the ensemble model. This method is effectively used in classification problems and provides better generalization performance.

train\_test\_split

train\_test\_split is a function in the model\_selection module of the scikit-learn library. This function is used to split the dataset into training and test subsets. It performs the splitting by randomly shuffling the data according to specified proportions. The separation of training and test data is important for evaluating the training performance of the model and measuring its generalization ability.

METRICS

accuracy\_score, precision\_score, recall\_score and f1\_score

accuracy\_score, precision\_score, recall\_score and f1\_score are classification performance metrics included in the metrics module of the scikit-learn library.

* accuracy\_score: Computes the ratio of correctly classified samples to the total number of samples. A high accuracy score indicates how well the classification model is performing.
* precision\_score: Calculates the ratio of true positives to the total predicted positives. It is useful when minimizing false positives is important. For example, in spam filtering, it aims to avoid incorrectly marking a legitimate email as spam.
* recall\_score: Calculates the ratio of true positives to the total actual positives. It is important when minimizing false negatives. For instance, it can be used in a medical diagnostic model that aims to correctly identify all diseased cases.
* f1\_score: Computes the harmonic mean of precision and recall values. It is a balanced performance measure. The F1 score is useful in situations where both minimizing false positives and false negatives are important.

A math equations with numbers

Description automatically generated

These metrics are used to evaluate the performance of classification models. They help assess the accuracy, false positive and false negative predictions, and the classification capabilities of the model.

ALGORITHMS USED IN THE PROJECT

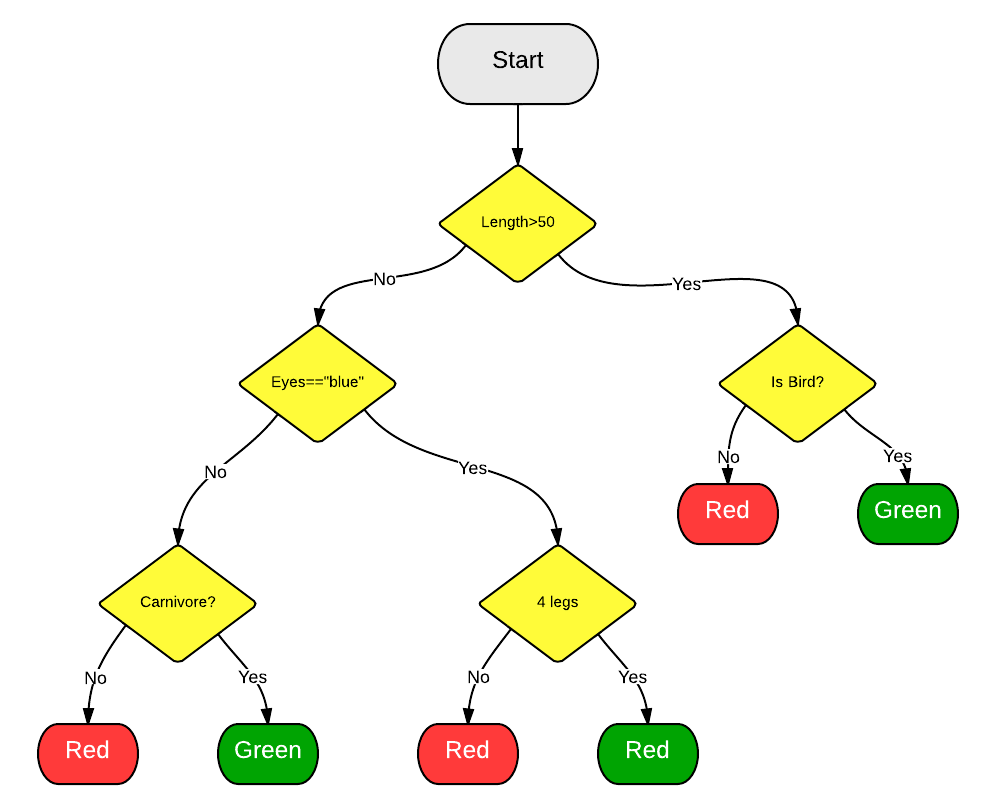
Random Forest Algorithm

Random Forest is one of the most popular and commonly used algorithms by Data Scientists. Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

Random forest is a versatile machine learning algorithm developed by Leo Breiman and Adele Cutler. It leverages an ensemble of multiple decision trees to generate predictions or classifications. By combining the outputs of these trees, the random forest algorithm delivers a consolidated and more accurate result.

Its widespread popularity stems from its user-friendly nature and adaptability, enabling it to tackle both classification and regression problems effectively. The algorithm’s strength lies in its ability to handle complex datasets and mitigate overfitting, making it a valuable tool for various predictive tasks in machine learning.

One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables, as in the case of regression, and categorical variables, as in the case of classification. It performs better for classification and regression tasks.



Using the Random Forest Algorithm in PyCharm

# Includes the necessary libraries  
import pandas as pd  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score  
  
# Include the dataset in the project  
data = pd.read\_csv('StudentsPerformance.csv')  
  
# Setting properties and target variable  
X = data.drop('gender', axis=1)  
y = data['gender']  
  
# Transform categorical attributes in a dataset into dummy variables  
X= pd.get\_dummies(X)  
  
# Splitting the dataset into training and test sets with train\_test\_split function  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Create a Random Forest classification model from the RandomForestClassifier class  
rf\_classifier = RandomForestClassifier()  
  
# Training rf\_classifier, a Random Forest classification model built using x\_train and y\_train datasets  
rf\_classifier.fit(X\_train, y\_train)  
  
# Making predictions for the X\_test dataset in the Random Forest classification model (rf\_classifier)  
# and assigning the prediction results to the y\_pred variable  
y\_pred = rf\_classifier.predict(X\_test)  
  
# The process of calculating the model's accuracy, precision, recall and F1 score  
accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred)  
recall = recall\_score(y\_test, y\_pred)  
f1 = f1\_score(y\_test, y\_pred)  
  
# Printing evaluation metrics  
print("Model Accuracy:", accuracy)  
print("Precision:", precision)  
print("Recall:", recall)  
print("F1 Score:", f1)

Output:

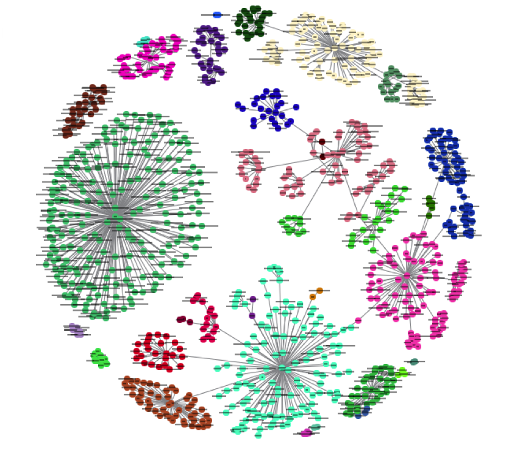
C:\Users\alpa\_\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\alpa\_\OneDrive\Masaüstü\PythonProject\RandomForest.py   
Model Accuracy: 0.87  
Precision: 0.8446601941747572  
Recall: 0.8969072164948454  
F1 Score: 0.87  
Process finished with exit code 0

K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a commonly used machine learning algorithm for classification and regression problems. This algorithm uses the nearest neighbors in the vicinity of a data point to classify it or make predictions. The KNN algorithm follows these steps:

* Data Preprocessing: Preprocessing steps such as handling missing values, scaling, and dealing with outliers are performed on the dataset.
* Determining the Number of Neighbors (K): K, a parameter that needs to be chosen for KNN, represents the number of nearest neighbors to consider when classifying a new data point.
* Distance Measurement: KNN calculates the distances between a new data point and its neighbors. Typically, the Euclidean distance is used, but other distance metrics can also be utilized.
* Training: During the training phase, the KNN algorithm learns and stores the dataset in memory.
* Prediction: To predict the class label of a new data point, the algorithm identifies its nearest neighbors and makes a prediction based on the majority vote. In regression problems, the prediction is based on the average of the nearest neighbors' values.

KNN is a straightforward and understandable algorithm commonly used for classification and regression tasks. It is sensitive to the structure and distribution of the dataset, so data preprocessing and scaling steps can significantly impact its performance. Additionally, choosing the right value for K is crucial as an incorrect K value can affect the model's performance. KNN is favored by many data scientists due to its simplicity and ease of implementation. However, it can have a high computational cost for large datasets and high-dimensional data.



Using K-Nearest Neighbors Algorithm in PyCharm

# Includes the necessary libraries  
import pandas as pd  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score  
  
# Include the dataset in the project  
data = pd.read\_csv('StudentsPerformance.csv')  
  
# Setting properties and target variable  
X = data.drop('lunch', axis=1)  
y = data['lunch']  
  
# Transform categorical attributes in a dataset into dummy variables  
X = pd.get\_dummies(X)  
  
# Splitting the dataset into training and test sets with train\_test\_split function  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Create a Random Forest classification model from the RandomForestClassifier class  
knn\_classifier = RandomForestClassifier()  
  
# Training knn\_classifier, a Random Forest classification model built using x\_train and y\_train datasets  
knn\_classifier.fit(X\_train, y\_train)  
  
# Making predictions for the X\_test dataset in the Random Forest classification model (knn\_classifier)  
# and assigning the prediction results to the y\_pred variable  
y\_pred = knn\_classifier.predict(X\_test)  
  
# The process of calculating the model's accuracy, precision, recall and F1 score  
accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred)  
recall = recall\_score(y\_test, y\_pred)  
f1 = f1\_score(y\_test, y\_pred)  
  
# Printing evaluation metrics  
print("Model Accuracy:", accuracy)  
print("Precision:", precision)  
print("Recall:", recall)  
print("F1 Score:", f1)

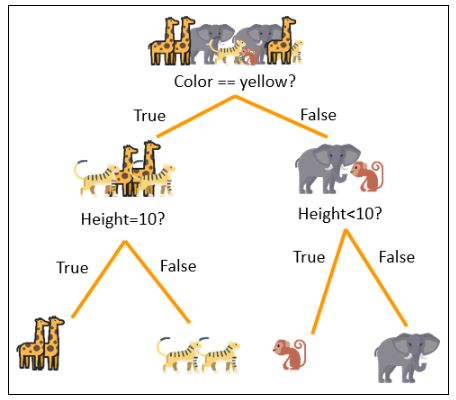
Output:

C:\Users\alpa\_\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\alpa\_\OneDrive\Masaüstü\PythonProject\DecisionTree.py  
Model Accuracy: 0.62  
Precision: 0.6717557251908397  
Recall: 0.7272727272727273  
F1 Score: 0.6984126984126985  
Process finished with exit code 0

Decision Tree Algorithm

A decision tree is a tree-based supervised learning method used to predict the output of a target variable. Supervised learning uses labeled data (data with known output variables) to make predictions with the help of regression and classification algorithms. Supervised learning algorithms act as a supervisor for training a model with a defined output variable. It learns from simple decision rules using the various data features. Decision trees in Python can be used to solve both classification and regression problems—they are frequently used in determining odds.

The following is an example of a simple decision tree used to classify different animals based on their features. We will be using the color and height of the animals as input features.



*Advantages of Using Decision Trees*

1. Decision trees are simple to understand, interpret, and visualize
2. They can effectively handle both numerical and categorical data
3. They can determine the worst, best, and expected values for several scenarios
4. Decision trees require little data preparation and data normalization
5. They perform well, even if the actual model violates the assumptions

Using Decision Tree Algorithm in PyCharm

# Includes the necessary libraries  
import pandas as pd  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score  
  
# Include the dataset in the project  
data = pd.read\_csv('StudentsPerformance.csv')  
  
# Setting properties and target variable  
X = data.drop('test preparation course', axis=1)  
y = data['test preparation course']  
  
# Transform categorical attributes in a dataset into dummy variables  
X = pd.get\_dummies(X)  
  
# Splitting the dataset into training and test sets with train\_test\_split function  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Training dt\_classifier, a Random Forest classification model built using x\_train and y\_train datasets  
dt\_classifier = DecisionTreeClassifier()  
  
# Training dt\_classifier, a Random Forest classification model built using x\_train and y\_train datasets  
dt\_classifier.fit(X\_train, y\_train)  
  
# Making predictions for the X\_test dataset in the Random Forest classification model (dt\_classifier)  
# and assigning the prediction results to the y\_pred variable  
y\_pred = dt\_classifier.predict(X\_test)  
  
# The process of calculating the model's accuracy, precision, recall and F1 score  
accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred)  
recall = recall\_score(y\_test, y\_pred)  
f1 = f1\_score(y\_test, y\_pred)  
  
# Printing evaluation metrics  
print("Model Accuracy:", accuracy)  
print("Precision:", precision)  
print("Recall:", recall)  
print("F1 Score:", f1)

Output:

C:\Users\alpa\_\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\alpa\_\OneDrive\Masaüstü\PythonProject\DecisionTree.py  
Model Accuracy: 0.62  
Precision: 0.6717557251908397  
Recall: 0.7272727272727273  
F1 Score: 0.6984126984126985  
Process finished with exit code 0

Final Version Of The Data Set Used In The Project

A screenshot of a computer

Description automatically generated

gender:

* female=1,
* male=2

race/ethnicity:

* group A=1,
* group B= 2
* group C=3,
* group D=4,
* group E=5

parental\_level\_of\_education:

* high school=1,
* some college=2,
* associate's degree=3,
* bachelor's degree=4,
* master's degree=5

lunch

* standard=1
* free/reduced=2

test preparation course

* none=1
* completed

REFERENCES:

Dataset:

* <https://www.kaggle.com/>
* <https://www.kaggle.com/datasets/spscientist/students-performance-in-exams>

Others:

1. <https://mode.com/python-tutorial/libraries/pandas/>
2. <https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/>
3. <https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>
4. <https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/>
5. <https://www.kdnuggets.com/2016/12/random-forests-python.html>
6. <https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/>
7. <https://www.simplilearn.com/tutorials/machine-learning-tutorial/decision-tree-in-python>