

Machine Learning and Related Application - CS LVL6 - S01 2023/2024

[Document Subtitle]

youssef mohamed 20200837

The presentation link from one drive : [Meeting in \_General\_-20231211\_045826-Meeting Recording.mp4](https://elsewedyedu1-my.sharepoint.com/:v:/g/personal/ya00837_tkh_edu_eg/ETOb_ceHh8BFlz6a7-7eZ9kB2MQP08vsXkiKnmUmQSf97Q?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=c6Yol4)

The code lines in GitHub: <https://github.com/abouhena/machine-learning-1/blob/3292f3729673ea5ab3611ac17f0900c835f1fd8e/machine22-2.ipynb>

First, of all I have chosen some data from the banks industry in which; The information relates to a Portuguese banking institution's phone-based direct marketing activities. Predicting whether a consumer will sign up for a term deposit is the classification target (variable y). Phone calls served as the basis for the marketing initiatives. It was frequently necessary to make multiple contacts with the same client to find out whether they would be subscribed to the product (bank term deposit) or not ('no'). Bank additional-full.csv, which is extremely similar to the data examined, has all cases (41188) and 20 inputs arranged chronologically (May 2008 to November 2010). To test more computationally intensive machine learning methods (like SVM). The smallest datasets are offered. Predicting whether a consumer will subscribe for a term deposit (yes/no) is the classification target (variable y). which the data collected a lot of information from the clients which is 21 columns and 41189 row.A screenshot of a computer

Description automatically generated

*bank-additional-full.csv*. (2021, January 7). Kaggle. <https://www.kaggle.com/datasets/sahistapatel96/bankadditionalfullcsv>

so I started to analysis and define my data A screenshot of a computer

Description automatically generated

Analyzing the clients data by a hostrogramA screen shot of a graph

Description automatically generated

The following plot sample builds a histogram that illustrates the age distribution of the dataset's clients using Matplotlib, a Python visualization tool. This visualization make of the 'age' column from the dataset. While plt.hist(dataset['age'], bins=20, color='skyblue', edgecolor='black') constructs the histogram with 20 bins, illustrating the frequency of different age ranges, plt.figure(figsize=(8, 6)) specifies the figure size to be presented. Plt.xlabel('Age') and plt.ylabel('Frequency') are used to add labels to represent the corresponding axes, while plt.title('Age Distribution of Clients') gives the plot a title. To improve readability, the plt.grid(True) method adds a grid. Lastly, the histogram is rendered for viewing and analysis.

I used a bar chart to show clients jobs averages

A graph of a number of people

Description automatically generated

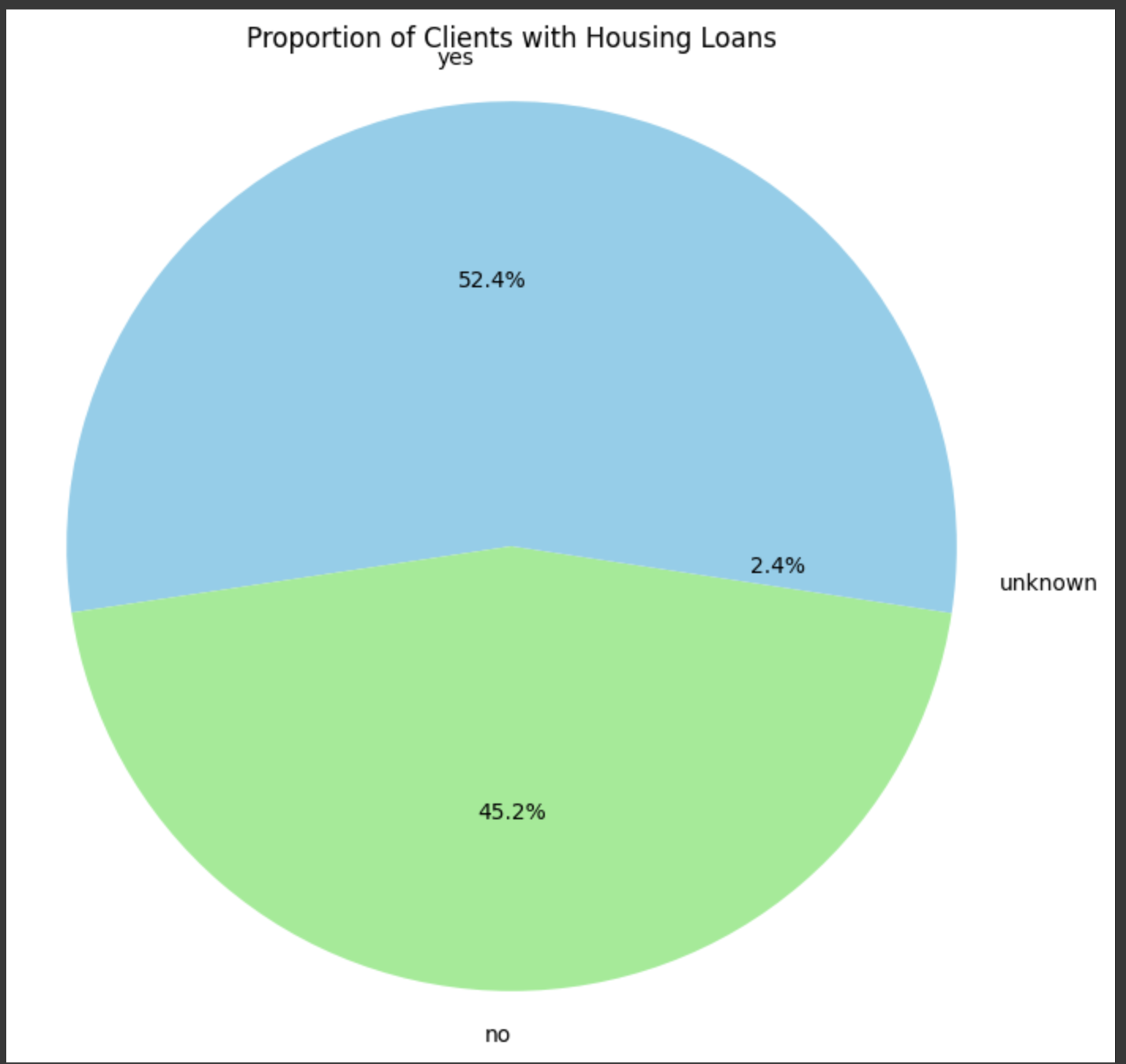
To visualise the distribution of jobs among clients in a dataset. First, the value counts of each job category among the customers are calculated for the 'job' column using the value\_counts() function in pandas. Matplotlib is then used to provide an understandable and informative bar plot illustration of this data. The generated graph shows the number of clients in various work categories. The height of each bar represents the number of clients linked with a particular job category. Each bar represents a separate job category. The graphic is color-coded for easier reading, the x-axis shows different employment types, and the y-axis indicates the count. This method of visualising data provides an easy way to understand how clients are distributed around various employment categories within the data.

A graph of numbers and letters

Description automatically generated with medium confidence

This grouped bar graph. collect line of code groups the dataset according to job categories as well as whether or not clients subscribed to a term deposit ('yes' or 'no'). Next, it uses Matplotlib to build a grouped bar chart that illustrates how these subscriptions differ for various job kinds. The number of "yes" responses (those who subscribed) and the number of "no" responses (those who didn't) are shown by the pairs of bars in each job category. Multiple job categories are displayed on the x-axis, the count is shown on the y-axis, and 'yes' and 'no' responses are indicated by different colours. This graphic makes it simple to analyse the subscription behaviour of various job types and determine which jobs are likely to have greater or lower term deposit subscription rates.

Pie Chart for the clients with house loans



I did count the values in the dataset's 'housing' column to determine the frequency of clients with and without housing loans. Next, it uses Matplotlib to generate a pie chart that shows the distribution of clients who have housing loans vs those who do not. Proportions are shown in the pie chart, where slices denote the proportion of clients with house loans ('yes') and those without ('no'). The autopct='%1.1f%%' argument formats the percentage display on the chart, and each slice has an appropriate title. The graphic displays the relative sizes of these two categories using colors—"skyblue" for "yes" and "lightgreen" for "no." By guaranteeing that the pie chart displays as a circle, the 'Equal aspect ratio' command makes it simpler to understand the percentage of clients who have home loans as opposed to those who do not. Also I wanted to know the missing values which were 2.4% for the unknown

I did a Bar graph to see if its balanced or not A screen shot of a computer

Description automatically generated

The 'y' column of the dataset is where this code first counts the instances of various classes. It then uses Matplotlib to create a bar plot so that the distribution of these classes can be shown. The heights of the bars correspond to the corresponding counts, and each bar indicates a class ('yes' or 'no').,'Skyblue' and'salmon' serve as plot-specific colour markers for the two classes,The distribution of classes in the dataset may be easily understood thanks to this visualisation. The code shows the counts of each class after the plot, giving a numerical depiction of the frequency of "yes" and "no" occurrences in the dataset. .Understanding the balance or imbalance between the two classes in the dataset is made easier with the help of this combined visual and mathematical representation.

**First Model: Random Forest**

A screenshot of a computer program

Description automatically generated

This code uses Python's scikit-learn module to create and assess a machine learning model to standard protocol. In order to transform categorical characteristics into a numerical format appropriate for modelling, it first divides the dataset into features and the target variable ('y'). The Random Forest Classifier is then trained by dividing the data into training and testing sets. The training data is used to train the classifier, which is set up with 100 trees. Based on the test features, the model then makes a prediction about the target variable. The accuracy of the predictions made on the test set is computed and shown to evaluate the model's performance and provide a measure of how effectively the model generalises to new data. The classification report also provides a thorough analysis of the model's recall, F1-score, precision, and support for every class in the target variable, providing information about how well the model performs in various contexts. With this method, creating, honing, and assessing a machine learning model for categorization tasks may be done methodically.

I did a confusion matrix for the random forest model A screenshot of a computer

Description automatically generated

First, the dataset is divided into categories based on predictive factors and the target variable ('y'). One-hot encoding is used to convert categorical data into a numerical representation that can be analysed, making model training easier. Then, to make sure the model learns patterns from one set and tests its performance on unseen data, the dataset is divided into separate training and testing subsets. The training data is used to create and train a Random Forest Classifier, which is configured with 100 trees and a fixed random state for reproducibility. Predictions on the test set are used to evaluate the predictive power of the model. The accuracy metric measures how well the model predicts the target variable overall. Moreover, a thorough classification report offers additional insights by displaying each class's precision, recall, F1-score, and support metrics class inside the target variable, allowing for a detailed evaluation of the model's effectiveness in several categories. A strong foundation for building, honing, and assessing machine learning models for categorization problems is provided by this methodical approach.

**Second Model: Decision Tree**

**A screenshot of a computer program

Description automatically generated**

This snippet of code uses the scikit-learn module in Python to carry out a machine learning workflow with a Decision Tree Classifier. The first step in the method is to divide the dataset into the target variable ('y') and features ('X'). One-hot encoding converts categorical characteristics into a numerical format that can be analysed, improving the model's capacity to interpret them. The train\_test\_split() function is then used to split the dataset into training and testing subsets, reserving 20% of the data for testing and guaranteeing the generalizability of the model. Using the training data, a Decision Tree Classifier with a default setup is trained to identify patterns in the features and target variable. Next, the test set's target variable is predicted by the trained model. In order to evaluate its efficiency, the code calculates and shows the accuracy score, which expresses how well the model predicts the target variable overall. Furthermore, an extensive report on classification is produced, which displays metrics such as precision, recall, F1-score, and support for every class in the target variable. This provides a thorough assessment of the model's effectiveness in several categories. This method demonstrates an organised process for building, honing, and assessing a machine learning model for decision tree-based categorization tasks.

**Confusion matrix for Decision Tree**

**A screenshot of a computer

Description automatically generated**

Based on the true labels (y\_test) and predicted labels (y\_pred\_dt) derived from a Decision Tree model, this code computes a confusion matrix. The **confusion matrix compares the expected and actual classifications to show how well the model performed. This matrix is computed by the scikit-learn confusion\_matrix() function.**

**After that, the code makes advantage of Seaborn's heatmap() function to show the confusion matrix graphically. The matrix is displayed on the heatmap, along with comments that show how many samples fall into each category. The heatmap's colours indicate how many cases were classified correctly and wrongly. The genuine labels are represented by the y-axis, and the anticipated labels are displayed on the x-axis. This graphic, which is frequently employed in assessments of classification models, aids in comprehending the accuracy and types of errors in the model. In this instance, it's particularly presenting the classification results of the Decision Tree model in several categories in an organised and easily understood format in order to demonstrate the model's performance.**

**After this two models I did a confusion matrix and a bar graph to compare both**

**A screenshot of a graph

Description automatically generated**

**A screen shot of a graph

Description automatically generated**

This code segment compares two classifiers: a Random Forest classifier and a Decision Tree classifier using Python's scikit-learn module. Both models are trained and accuracy tested using test data after the dataset has been prepped by encoding categorical characteristics and divided into training and testing sets. The performance of the Random Forest Classifier and Decision Tree Classifier may be directly compared thanks to the computation and visualisation of their accuracies using a bar graph. The accuracy of each model is briefly displayed in this visualisation, making it easier to determine which model works best for the particular dataset.

**Third Model: Artificial Neural Network (ANN)**

**A screenshot of a computer program

Description automatically generated**

This code implements a function called build\_ann\_model() that builds artificial neural networks (ANNs) for binary classification tasks using TensorFlow and Keras. It first preprocesses the data by removing rows that contain missing values and dividing the target variable from the features. Label encoding for the goal and one-hot encoding for features are used to encode categorical variables. Train\_test\_split() is used to divide the dataset into training and testing sets. Three tightly linked layers make up the architecture of the ANN model: an output layer that uses the sigmoid activation function for binary classification, and two hidden layers with 128 and 64 neurons, respectively, using the ReLU activation function. The binary cross-entropy loss function and Adam optimizer are used to compile the model. After that, it is trained for ten epochs using the training set a validation split to assess its performance, with a batch size of 32. Lastly, it prints the accuracy score after assessing the trained model's accuracy on the test set. The process of creating, honing, and testing an ANN model for binary classification tasks is encapsulated in this function.

**Confusion matrix for ANN**

**A screenshot of a computer

Description automatically generated**

The purpose of this snippet of code is to assess an Artificial Neural Network (ANN) model's performance on a binary classification test. First, it uses a predetermined threshold (e.g., 0.5) to convert the projected probabilities from the ANN model into binary predictions. By determining whether the anticipated probability are greater than the threshold, these binary predictions are obtained. In order to compare categorical labels in the test set in the confusion matrix, it simultaneously converts them to binary labels (1 for "yes" and 0 for other categories). It then builds a confusion matrix by comparing the predicted and real labels using scikit-learn's confusion\_matrix() function to illustrate how well the model performed. Lastly, a Seaborn heatmap visualisation offers an understandable visual depiction of the confusion matrix, assisting in the assessment of the model's accuracy and highlighting accurate and inaccurate classifications, as well as pointing out instances in which the model performs well or in which it has to be improved in order to better anticipate outcomes for the given binary classification job.

**ROC & AUC Curve for ANN**

**A screen shot of a graph

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

This code snippet focuses on calculating the Area Under the Curve (AUC) and using a Receiver Operating Characteristic (ROC) curve to evaluate the effectiveness of an Artificial Neural Network (ANN) model. Based on the true labels (y\_test) and projected probabilities (y\_pred\_ann) from the model, the code computes the False Positive Rate (fpr), True Positive Rate (tpr), and corresponding thresholds using scikit-learn's roc\_curve() function. The ROC curve is then integrated to calculate the AUC, a measure of the model's discriminative power. Next, using Matplotlib, the code creates a graphical depiction of the ROC curve, with the False Positive Rate on the x-axis and the True Positive Rate on the y-axis, representing the model's performance. Furthermore, a diagonal line that shows the effectiveness of a random classifier is provided for reference. The AUC value and the ROC curve visualisation offer important insights into the model's capacity to discriminate across classes; a higher AUC indicates superior performance in classification tasks.

**ROC & AUC Curve for Random Forest, Decision Tree, and ANN**

A screen shot of a graph

Description automatically generated

This plot uses Receiver Operating Characteristic (ROC) curves and Area Under the Curve (AUC) metrics to assess and compare the performance of three different models: Decision Tree, Random Forest, and Artificial Neural Network. It calculates the ROC curves and AUC values for each model based on true labels and expected probabilities by utilising scikit-learn's functionality. These ROC curves are shown in the following Matplotlib visualisation, which compares the False Positive Rates against the True Positive Rates. The models' discriminatory abilities can be directly compared thanks to the labelled curves and various colours, which makes it easier to determine which model works best for classification tasks. The performance of a random classifier is represented by the diagonal dashed line, which gives the model performances context. This kind of study provides an unambiguous visual evaluation, enabling to make well-informed selections about which model—shown by higher AUC values—performs best at class distinction.

# -\*- coding: utf-8 -\*-

"""machine22.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/10Rzaz8jAe4a2ChGa31w6C\_M9foGj\_DcQ

Importing Libraries

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

dataset = pd.read\_csv('/content/bank-additional-full.csv',sep=';')

dataset

"""Defining Data"""

dataset.dtypes

dataset.info()

dataset.shape

dataset.columns

dataset.isnull().sum()

"""Analysisng Data

Histogram

"""

plt.figure(figsize=(8, 6))

plt.hist(dataset['age'], bins=20, color='skyblue', edgecolor='black')

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.title('Age Distribution of Clients')

plt.grid(True)

plt.show()

"""In this Histogram i wanted to show the age frequency of the clinets

Bar Chart

"""

job\_counts = dataset['job'].value\_counts()

plt.figure(figsize=(10, 6))

job\_counts.plot(kind='bar', color='skyblue', edgecolor='black')

plt.xlabel('Job Category')

plt.ylabel('Count')

plt.title('Count of Clients in Different Job Categories')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

"""i wanted to show the count of clients in each job categories

Pie Chart

"""

# Count the occurrences of clients with and without housing loans

housing\_loan\_counts = dataset['housing'].value\_counts()

# Plotting the pie chart for housing loan distribution

plt.figure(figsize=(8, 8))

plt.pie(housing\_loan\_counts, labels=housing\_loan\_counts.index, autopct='%1.1f%%', colors=['skyblue', 'lightgreen'])

plt.title('Proportion of Clients with Housing Loans')

plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()

"""i wanted to show the propotion of clients with and without house loans and also showing the missing data with 2.4%

Heat Map

"""

numerical\_columns = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',

'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']

# Calculate correlation matrix

correlation\_matrix = dataset[numerical\_columns].corr()

# Plotting the heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Correlation Matrix of Numerical Variables')

plt.show()

"""heatmap showing the correlation matrix between numerical columns like age, duration, campaign,etc.

Count Plot

"""

plt.figure(figsize=(8, 6))

sns.countplot(data=dataset, x='day\_of\_week', palette='viridis')

plt.xlabel('Day of the Week')

plt.ylabel('Count of Contacts')

plt.title('Count of Contacts on Different Days of the Week')

plt.show()

"""countplot showing the number of contacts made on each day of the week during the campaign.

Categorical vs. Numerical

violin plot

> Boxplots or Violin Plots: Visualize the distribution of a numerical column concerning the 'y' categories.

"""

plt.figure(figsize=(8, 6))

sns.violinplot(data=dataset, x='y', y='age', palette='coolwarm')

plt.xlabel('Subscription to Term Deposit (y)')

plt.ylabel('Age')

plt.title('Age Distribution vs. Subscription to Term Deposit')

plt.show()

"""violin plot representing the distribution of ages for clients who either subscribed ('yes') or didn't subscribe ('no') to a term deposit. It helps to visually understand if there's a significant difference in age distribution between those who subscribed and those who didn't. The width of the plot at various age levels represents the density or number of individuals in each category.

Bar Charts with Means or Medians

> Show the mean or median of a numerical column for each 'y' category

"""

mean\_duration = dataset.groupby('y')['duration'].mean().reset\_index()

# Plotting a bar chart for mean duration by 'y' category

plt.figure(figsize=(8, 6))

sns.barplot(data=mean\_duration, x='y', y='duration', palette='coolwarm', ci=None)

plt.xlabel('Subscription to Term Deposit (y)')

plt.ylabel('Mean Duration')

plt.title('Mean Duration vs. Subscription to Term Deposit')

plt.show()

"""Median"""

median\_duration = dataset.groupby('y')['duration'].median().reset\_index()

# Plotting a bar chart for median duration by 'y' category

plt.figure(figsize=(8, 6))

sns.barplot(data=median\_duration, x='y', y='duration', palette='coolwarm', ci=None)

plt.xlabel('Subscription to Term Deposit (y)')

plt.ylabel('Median Duration')

plt.title('Median Duration vs. Subscription to Term Deposit')

plt.show()

"""Categorical vs. Categorical

Countplots

> : Show the count of 'y' categories across other categorical columns.

"""

# Plotting a countplot for 'y' categories across the 'marital' column

plt.figure(figsize=(8, 6))

sns.countplot(data=dataset, x='marital', hue='y', palette='coolwarm')

plt.xlabel('Marital Status')

plt.ylabel('Count')

plt.title('Count of "y" Categories Across Marital Status')

plt.legend(title='Subscription to Term Deposit', loc='upper right')

plt.show()

"""countplot showing the count of 'yes' and 'no' categories for the 'y' variable across different marital statuses. Each marital status category will have bars indicating the count of 'yes' and 'no' instances

Grouped Bar Charts

>Show the count of 'y' categories grouped by another categorical column.

"""

# Grouping data by 'job' and 'y', calculating counts

grouped\_data = dataset.groupby(['job', 'y']).size().unstack()

# Plotting a grouped bar chart for 'y' categories across 'job'

plt.figure(figsize=(12, 6))

grouped\_data.plot(kind='bar', color=['#1f77b4', '#ff7f0e'], figsize=(12, 6))

plt.xlabel('Job Category')

plt.ylabel('Count')

plt.title('Count of "y" Categories Grouped by Job')

plt.legend(title='Subscription to Term Deposit', loc='upper right')

plt.show()

"""grouped bar chart showing the count of 'yes' and 'no' categories within each job category. Each job category will have two adjacent bars representing the count of 'yes' and 'no' instances of the 'y' variable.

Numerical vs. Numerical

Scatterplots

>Explore the relationship between two numerical columns, differentiating by 'y' categories using color or marker.

"""

# Plotting a scatterplot for 'duration' vs. 'campaign', differentiated by 'y' categories

plt.figure(figsize=(8, 6))

sns.scatterplot(data=dataset, x='duration', y='campaign', hue='y', palette='coolwarm')

plt.xlabel('Duration')

plt.ylabel('Campaign')

plt.title('Relationship between Duration and Campaign, Differentiated by Subscription to Term Deposit')

plt.legend(title='Subscription to Term Deposit', loc='upper right')

plt.show()

"""scatterplot with 'duration' on the x-axis and 'campaign' on the y-axis. The points will be differentiated by color, representing the 'yes' and 'no' categories of the 'y' variable.

Time Series Analysis

Line Plots

> Show trends of 'y' categories over time if there's a time-related column.

"""

# Plotting trends of 'y' categories over days of the week ('day\_of\_week')

plt.figure(figsize=(10, 6))

sns.countplot(data=dataset, x='day\_of\_week', hue='y', palette='coolwarm')

plt.xlabel('Day of Week')

plt.ylabel('Count')

plt.title('Subscription to Term Deposit over Days of the Week')

plt.legend(title='Subscription to Term Deposit', loc='upper right')

plt.tight\_layout()

plt.show()

"""countplot showing the trends of 'y' categories ('yes' and 'no') over different days of the week

Random Forest model

Bar plot

> to see if its balance or not

"""

class\_counts = data['y'].value\_counts()

# Plotting the distribution of classes

plt.figure(figsize=(6, 4))

class\_counts.plot(kind='bar', color=['skyblue', 'salmon'])

plt.xlabel('Classes')

plt.ylabel('Count')

plt.title('Distribution of Classes')

plt.show()

# Displaying the class counts

print("Class Counts:")

print(class\_counts)

pip install tensorflow

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.preprocessing import LabelEncoder, StandardScaler

from keras.models import Sequential

from keras.layers import Dense

# Load your CSV file into a DataFrame

data = pd.read\_csv('/content/bank-additional-full.csv', sep=';')

# Assuming 'y' is the target variable, and other columns are features

# Selecting features and target variable

X = data.drop('y', axis=1) # Features

y = data['y'] # Target variable

"""Random Forest"""

# Assuming 'y' is the target variable, and other columns are features

# Selecting features and target variable

X = data.drop('y', axis=1) # Features

y = data['y'] # Target variable

# Convert categorical variables to numerical using one-hot encoding

X = pd.get\_dummies(X)

# Splitting the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initializing the Random Forest Classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Training the model

rf\_classifier.fit(X\_train, y\_train)

# Making predictions on the test set

y\_pred = rf\_classifier.predict(X\_test)

# Evaluating the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Assuming you have the true labels (y\_true) and predicted labels (y\_pred\_rf) for the Random Forest model

# Creating the confusion matrix

conf\_matrix\_rf = confusion\_matrix(y\_test, y\_pred\_rf)

# Displaying the confusion matrix using a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_rf, annot=True, cmap='Greens', fmt='d')

plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix for Random Forest Model')

plt.show()

"""Decision Tree model"""

# Assuming 'y' is the target variable, and other columns are features

# Selecting features and target variable

X = data.drop('y', axis=1) # Features

y = data['y'] # Target variable

# Convert categorical variables to numerical using one-hot encoding

X = pd.get\_dummies(X)

# Splitting the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initializing the Decision Tree Classifier

dt\_classifier = DecisionTreeClassifier(random\_state=42)

# Training the model

dt\_classifier.fit(X\_train, y\_train)

# Making predictions on the test set

y\_pred = dt\_classifier.predict(X\_test)

# Evaluating the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

# Assuming you have the true labels (y\_true) and predicted labels (y\_pred\_dt) for the Decision Tree model

# Creating the confusion matrix

conf\_matrix\_dt = confusion\_matrix(y\_test, y\_pred\_dt)

# Displaying the confusion matrix using a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_dt, annot=True, cmap='Blues', fmt='d')

plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix for Decision Tree Model')

plt.show()

"""confusion matrix"""

X = data.drop('y', axis=1) # Features

y = data['y'] # Target variable

# Convert categorical variables to numerical using one-hot encoding

X = pd.get\_dummies(X)

# Splitting the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initializing the Decision Tree Classifier

dt\_classifier = DecisionTreeClassifier(random\_state=42)

# Training the Decision Tree model

dt\_classifier.fit(X\_train, y\_train)

# Making predictions on the test set with Decision Tree

y\_pred\_dt = dt\_classifier.predict(X\_test)

# Creating a confusion matrix for Decision Tree

cm\_dt = confusion\_matrix(y\_test, y\_pred\_dt)

# Plotting the confusion matrix for Decision Tree

plt.figure(figsize=(6, 4))

sns.heatmap(cm\_dt, annot=True, cmap='Blues', fmt='d', cbar=False)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix for Decision Tree')

plt.show()

# Initializing the Random Forest Classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Training the Random Forest model

rf\_classifier.fit(X\_train, y\_train)

# Making predictions on the test set with Random Forest

y\_pred\_rf = rf\_classifier.predict(X\_test)

# Creating a confusion matrix for Random Forest

cm\_rf = confusion\_matrix(y\_test, y\_pred\_rf)

# Plotting the confusion matrix for Random Forest

plt.figure(figsize=(6, 4))

sns.heatmap(cm\_rf, annot=True, cmap='Greens', fmt='d', cbar=False)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix for Random Forest')

plt.show()

# Assuming 'y' is the target variable, and other columns are features

# Selecting features and target variable

X = data.drop('y', axis=1) # Features

y = data['y'] # Target variable

# Convert categorical variables to numerical using one-hot encoding

X = pd.get\_dummies(X)

# Splitting the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initializing the Decision Tree Classifier

dt\_classifier = DecisionTreeClassifier(random\_state=42)

# Training the Decision Tree model

dt\_classifier.fit(X\_train, y\_train)

# Making predictions on the test set with Decision Tree

y\_pred\_dt = dt\_classifier.predict(X\_test)

# Calculating accuracy of Decision Tree

accuracy\_dt = accuracy\_score(y\_test, y\_pred\_dt)

# Initializing the Random Forest Classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Training the Random Forest model

rf\_classifier.fit(X\_train, y\_train)

# Making predictions on the test set with Random Forest

y\_pred\_rf = rf\_classifier.predict(X\_test)

# Calculating accuracy of Random Forest

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

# Creating a bar graph to compare accuracies

models = ['Decision Tree', 'Random Forest']

accuracies = [accuracy\_dt, accuracy\_rf]

plt.figure(figsize=(8, 5))

plt.bar(models, accuracies, color=['blue', 'green'])

plt.ylabel('Accuracy')

plt.title('Accuracy Comparison between Decision Tree and Random Forest')

plt.ylim(0.85, 1) # Adjust ylim if needed

plt.show()

"""Artificial Neural Network (ANN)"""

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

def build\_ann\_model(data):

# Drop rows with missing values if any

data = data.dropna()

# Select features (X) and target (y)

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

y = data['y']

# Convert categorical variables to numerical using Label Encoding

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

# One-hot encode categorical variables in X

X = pd.get\_dummies(X)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Build the ANN model

model = Sequential()

model.add(Dense(units=128, activation='relu', input\_dim=X\_train.shape[1]))

model.add(Dense(units=64, activation='relu'))

model.add(Dense(units=1, activation='sigmoid')) # Output layer for binary classification

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.1)

# Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Accuracy: {accuracy}')

return model

# Assuming 'data' is your dataset DataFrame

# Call the function to build and train the ANN model

trained\_ann\_model = build\_ann\_model(data)

# Assuming you have a trained ANN model named 'trained\_ann\_model' and test data 'X\_test'

# Generate predictions for the test set using the ANN model

y\_pred\_ann = trained\_ann\_model.predict(X\_test)

# Ensure y\_pred\_ann is a one-dimensional array containing predicted probabilities for the positive class

# For instance, if it's a binary classification task, use y\_pred\_ann[:, 1] for the positive class

# Now, you can proceed to use y\_pred\_ann in the AUC curve calculation

from sklearn.metrics import roc\_curve, auc

# Assuming 'y\_test' contains the true labels for the test set

# Calculate the fpr and tpr for all thresholds using predicted probabilities

fpr\_ann, tpr\_ann, thresholds = roc\_curve(y\_true\_dummy, y\_pred\_ann)

auc\_ann = auc(fpr\_ann, tpr\_ann)

# Print the AUC score

print(f"AUC for ANN: {auc\_ann}")

"""ANN module using TensorFlow/Keras for the dataset. It includes functions for data preprocessing, model creation, training, and evaluation

"""

# Convert probabilities to binary predictions using a threshold (e.g., 0.5)

threshold = 0.5

y\_pred\_binary = (y\_pred\_ann > threshold).astype(int)

# Convert categorical labels to binary labels

y\_test\_binary = (y\_test == 'yes').astype(int)

# Create the confusion matrix

conf\_matrix\_ann = confusion\_matrix(y\_test\_binary, y\_pred\_binary)

# Display the confusion matrix using a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_ann, annot=True, cmap='YlGnBu', fmt='d')

plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix for ANN Model')

plt.show()

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# Assuming 'y\_test' contains the true labels for the test set and 'y\_pred\_ann' contains predicted probabilities

# Calculate the fpr and tpr for all thresholds using predicted probabilities

fpr\_ann, tpr\_ann, \_ = roc\_curve(y\_true\_dummy, y\_pred\_ann)

# Calculate the AUC (Area Under the Curve) for the ANN model

auc\_ann = auc(fpr\_ann, tpr\_ann)

# Plotting the ROC curve

plt.figure(figsize=(8, 6))

plt.plot(fpr\_ann, tpr\_ann, color='orange', lw=2, label=f'ANN (AUC = {auc\_ann:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--') # Diagonal line for random classifier

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve for ANN')

plt.legend(loc='lower right')

plt.show()

from sklearn.preprocessing import label\_binarize

# Binarize the true labels if they are not binary yet (necessary for multi-class problems)

# Replace 'n\_classes' with the number of classes in your dataset

n\_classes = 2 # Change this according to your dataset

y\_test\_bin = label\_binarize(y\_test, classes=np.arange(n\_classes))

# Assuming 'y\_pred\_ann' contains predicted probabilities for each class

# Convert predicted probabilities to predicted labels (binary)

threshold = 0.5 # Adjust threshold if needed

y\_pred\_labels = np.where(y\_pred\_ann >= threshold, 1, 0)

# Create the confusion matrix using binary labels

conf\_matrix\_ann = confusion\_matrix(y\_test\_bin, y\_pred\_labels)

from sklearn.metrics import accuracy\_score

# Decision Tree model

if 'DecisionTreeClassifier' in locals():

dt\_model = DecisionTreeClassifier()

y\_pred\_dt = dt\_model.fit(X\_train, y\_train).predict(X\_test)

accuracy\_dt = accuracy\_score(y\_test, y\_pred\_dt) # Calculate accuracy

print(f"Accuracy of Decision Tree: {accuracy\_dt:.4f}")

else:

print("Decision Tree model not found in variables.")

# Random Forest model

if 'RandomForestClassifier' in locals():

rf\_model = RandomForestClassifier()

y\_pred\_rf = rf\_model.fit(X\_train, y\_train).predict(X\_test)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf) # Calculate accuracy

print(f"Accuracy of Random Forest: {accuracy\_rf:.4f}")

else:

print("Random Forest model not found in variables.")

# Mapping string predictions to numerical labels

y\_pred\_dt\_numeric = np.where(y\_pred\_dt == 'yes', 1, 0)

# Compute ROC curve and AUC for Decision Tree

fpr\_dt, tpr\_dt, \_ = roc\_curve(y\_test\_binary, y\_pred\_dt\_numeric)

auc\_dt = auc(fpr\_dt, tpr\_dt)

# Convert 'yes'/'no' to numerical labels in y\_pred\_rf

label\_encoder = LabelEncoder()

y\_pred\_rf\_numeric = label\_encoder.fit\_transform(y\_pred\_rf)

# Ensure the labels are binary (0 and 1)

y\_pred\_rf\_numeric = np.where(y\_pred\_rf\_numeric == 1, 1, 0)

# Compute ROC curve and AUC for Random Forest

fpr\_rf, tpr\_rf, \_ = roc\_curve(y\_test\_binary, y\_pred\_rf\_numeric)

auc\_rf = auc(fpr\_rf, tpr\_rf)

# Map string predictions to numerical labels

y\_pred\_rf\_numeric = np.where(y\_pred\_rf == 'yes', 1, 0)

# Compute ROC curve and AUC for Random Forest

fpr\_rf, tpr\_rf, \_ = roc\_curve(y\_test\_binary, y\_pred\_rf\_numeric)

auc\_rf = auc(fpr\_rf, tpr\_rf)

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# Assuming you have y\_pred\_dt and y\_pred\_rf available

# Create artificial labels for illustration purposes (replace this with actual labels if available)

import numpy as np

y\_true\_dummy = np.random.randint(2, size=len(y\_pred\_dt)) # Create dummy labels for illustration

# Compute ROC curve and AUC for Decision Tree

fpr\_dt, tpr\_dt, \_ = roc\_curve(y\_test\_binary, y\_pred\_dt\_numeric)

auc\_dt = auc(fpr\_dt, tpr\_dt)

# Compute ROC curve and AUC for Random Forest

fpr\_rf, tpr\_rf, \_ = roc\_curve(y\_test\_binary, y\_pred\_rf\_numeric)

auc\_rf = auc(fpr\_rf, tpr\_rf)

# Plot ROC curves for both models

plt.figure(figsize=(8, 6))

plt.plot(fpr\_dt, tpr\_dt, color='blue', lw=2, label=f'Decision Tree (AUC = {auc\_dt:.2f})')

plt.plot(fpr\_rf, tpr\_rf, color='red', lw=2, label=f'Random Forest (AUC = {auc\_rf:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--') # Diagonal line for random classifier

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve (Decision Tree vs Random Forest)')

plt.legend(loc='lower right')

plt.show()

print(f"Shape of y\_test\_binary: {y\_test\_binary.shape}")

print(f"Shape of y\_pred\_rf: {y\_pred\_rf.shape}")

print(f"Unique values in y\_test\_binary: {np.unique(y\_test\_binary)}")

print(f"Unique values in y\_pred\_rf: {np.unique(y\_pred\_rf)}")

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# Assuming you have the true labels (y\_true) and predicted probabilities for each model

# Replace y\_pred\_dt, y\_pred\_rf, y\_pred\_ann with actual predicted probabilities for each model

# Compute ROC curve and AUC for Decision Tree

fpr\_dt, tpr\_dt, \_ = roc\_curve(y\_test\_binary, y\_pred\_dt\_numeric)

auc\_dt = auc(fpr\_dt, tpr\_dt)

# Compute ROC curve and AUC for Random Forest

fpr\_rf, tpr\_rf, \_ = roc\_curve(y\_test\_binary, y\_pred\_rf\_numeric)

auc\_rf = auc(fpr\_rf, tpr\_rf)

# Compute ROC curve and AUC for ANN

fpr\_ann, tpr\_ann, \_ = roc\_curve(y\_true\_dummy, y\_pred\_ann)

auc\_ann = auc(fpr\_ann, tpr\_ann)

# Plot ROC curves for all three models with custom labels

plt.figure(figsize=(8, 6))

plt.plot(fpr\_dt, tpr\_dt, color='blue', lw=2, label=f'Decision Tree (AUC = {auc\_dt:.2f})')

plt.plot(fpr\_rf, tpr\_rf, color='red', lw=2, label=f'Random Forest (AUC = {auc\_rf:.2f})')

plt.plot(fpr\_ann, tpr\_ann, color='green', lw=2, label=f'ANN (AUC = {auc\_ann:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--') # Diagonal line for random classifier

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve for Three Models')

plt.legend(loc='lower right')

plt.show()

print(classification\_report(y\_test, y\_pred))