|  |
| --- |
| **Practical Assignment 1:  Linear and Logistic Regression**  **1. Create ‘sales’ Data set having 5 columns namely: ID, TV, Radio, Newspaper and Sales.(random**  **500 entries) Build a linear regression model by identifying independent and target variables. Split the variables into training and testing sets. then divide the training and testing sets into a 7:3 ratio,**  **respectively and print them. Build a simple linear regression model.**  import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import r2\_score, mean\_squared\_error  import matplotlib.pyplot as plt  # 1. Create dataset  np.random.seed(42)  data = pd.DataFrame({  'ID': np.arange(1, 501),  'flat': np.random.randint(1, 100, 500), # Flats available  'houses': np.random.randint(1, 50, 500), # Houses available  })  # 2. Generate 'purchases' with some correlation  data['purchases'] = (  1.5 \* data['flat'] +  2.8 \* data['houses'] +  np.random.normal(0, 10, 500) # Noise  )  # 3. Define independent (X) and target (y) variables  X = data[['flat', 'houses']]  y = data['purchases']  # 4. Split dataset into 70% training and 30% testing  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)  # 5. Print training and testing sets  print("Training Features (X\_train):\n", X\_train.head(), "\n")  print("Training Target (y\_train):\n", y\_train.head(), "\n")  print("Testing Features (X\_test):\n", X\_test.head(), "\n")  print("Testing Target (y\_test):\n", y\_test.head(), "\n")  # 6. Build and train linear regression model  model = LinearRegression()  model.fit(X\_train, y\_train)  # 7. Print model details  print("\nIntercept:", model.intercept\_)  print("Coefficients (flat, houses):", model.coef\_)  # 8. Predict on test set  y\_pred = model.predict(X\_test)  # 9. Evaluate model performance  r2 = r2\_score(y\_test, y\_pred)  rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  print(f"\nR² Score: {r2:.4f}")  print(f"RMSE: {rmse:.4f}")  # 10. Visualization: Actual vs Predicted Purchases  plt.figure(figsize=(8,6))  plt.scatter(y\_test, y\_pred, color='blue', alpha=0.6)  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linewidth=2)  plt.title("Actual vs Predicted Purchases")  plt.xlabel("Actual Purchases")  plt.ylabel("Predicted Purchases")  plt.grid(True)  plt.tight\_layout()  plt.show()  **2. Create ‘real\_estate’ Data set having 4 columns namely: ID,flat, houses and purchases (random 500**  **entries). Build a linear regression model by identifying independent and target variable. Split the**  **variables into training and testing sets and print them. Build a simple linear regression model for**  **predicting purchases.**  import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import r2\_score, mean\_squared\_error  import matplotlib.pyplot as plt  # 1. Create dataset  np.random.seed(42)  data = pd.DataFrame({  'ID': np.arange(1, 501),  'flat': np.random.randint(1, 100, 500), # Flats available  'houses': np.random.randint(1, 50, 500), # Houses available  })  # 2. Generate 'purchases' with some correlation  data['purchases'] = (  1.5 \* data['flat'] +  2.8 \* data['houses'] +  np.random.normal(0, 10, 500) # Noise  )  # 3. Define independent (X) and target (y) variables  X = data[['flat', 'houses']]  y = data['purchases']  # 4. Split dataset into 70% training and 30% testing  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)  # 5. Print training and testing sets  print("Training Features (X\_train):\n", X\_train.head(), "\n")  print("Training Target (y\_train):\n", y\_train.head(), "\n")  print("Testing Features (X\_test):\n", X\_test.head(), "\n")  print("Testing Target (y\_test):\n", y\_test.head(), "\n")  # 6. Build and train linear regression model  model = LinearRegression()  model.fit(X\_train, y\_train)  # 7. Print model details  print("\nIntercept:", model.intercept\_)  print("Coefficients (flat, houses):", model.coef\_)  # 8. Predict on test set  y\_pred = model.predict(X\_test)  # 9. Evaluate model performance  r2 = r2\_score(y\_test, y\_pred)  rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  print(f"\nR² Score: {r2:.4f}")  print(f"RMSE: {rmse:.4f}")  # 10. Visualization: Actual vs Predicted Purchases  plt.figure(figsize=(8,6))  plt.scatter(y\_test, y\_pred, color='blue', alpha=0.6)  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linewidth=2)  plt.title("Actual vs Predicted Purchases")  plt.xlabel("Actual Purchases")  plt.ylabel("Predicted Purchases")  plt.grid(True)  plt.tight\_layout()  plt.show()  **3. Build a simple linear regression model for Fish Species Weight Prediction. (download dataset**  [**https://www.kaggle.com/aungpyaeap/fish-market?select=Fish.csv**](https://www.kaggle.com/aungpyaeap/fish-market?select=Fish.csv)  import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import r2\_score, mean\_squared\_error  import matplotlib.pyplot as plt  import numpy as np  # 1. Load dataset  # Replace 'Fish.csv' with your local path or the raw URL  data = pd.read\_csv('Fish.csv')  # 2. Select feature and target  X = data[['Length3']] # Independent variable  y = data['Weight'] # Target variable  # 3. Split into training (70%) and testing (30%) sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X, y, test\_size=0.3, random\_state=42)  # 4. Print sample data  print("Training feature sample:\n", X\_train.head(), "\n")  print("Training target sample:\n", y\_train.head(), "\n")  print("Testing feature sample:\n", X\_test.head(), "\n")  print("Testing target sample:\n", y\_test.head(), "\n")  # 5. Build and train a simple linear regression model  model = LinearRegression()  model.fit(X\_train, y\_train)  # 6. Model output  print("Intercept:", model.intercept\_)  print("Coefficient for Length3:", model.coef\_[0])  # 7. Make predictions on test set  y\_pred = model.predict(X\_test)  # 8. Evaluate model performance  r2 = r2\_score(y\_test, y\_pred)  rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  print(f"R² Score: {r2:.4f}")  print(f"RMSE: {rmse:.4f}")  # 9. Visualize actual vs predicted weight  plt.figure(figsize=(8, 6))  plt.scatter(X\_test, y\_test, color='blue', alpha=0.6, label='Actual weight')  plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Prediction')  plt.title("Fish Weight Prediction: Actual vs Predicted (Length3)")  plt.xlabel("Length3")  plt.ylabel("Weight")  plt.legend()  plt.grid(True)  plt.tight\_layout()  plt.show()  **Practical Assignment 2:**  **Frequent itemset and Association rule mining**  **1. Download the Market basket dataset.**  **Write a python program to read the dataset and dis play its information.**  **Preprocess the data (drop null values etc.)**  **Convert the categorical values into numeric format.**  **Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules.**  # Import necessary libraries  import pandas as pd  from mlxtend.frequent\_patterns import apriori, association\_rules  # Step 1: Load the dataset  # Replace 'Market\_Basket.csv' with your file path  data = pd.read\_csv("basket.csv")  # Step 2: Display dataset information  print("Dataset Information:")  print(data.info())  print("\nFirst 5 Rows:")  print(data.head())  # Step 3: Preprocess data  # Drop null values  data.dropna(inplace=True)  # Step 4: Convert categorical values into numeric format  # Convert transaction data into one-hot encoding  basket = pd.get\_dummies(data)  print("\nAfter Encoding:")  print(basket.head())  # Step 5: Apply Apriori Algorithm  # Generate frequent itemsets with minimum support  frequent\_itemsets = apriori(basket, min\_support=0.05, use\_colnames=True)  print("\nFrequent Itemsets:")  print(frequent\_itemsets)  # Step 6: Generate association rules  rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1.0)  print("\nAssociation Rules:")  print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']]) |
| **Practical Assignment 3 :   Text and Social Media Analytics**   1. **Consider any text paragraph. Preprocess the text to remove any special characters and digits. Generate the summary using the extractive summarization process.**   import re  from nltk.tokenize import sent\_tokenize, word\_tokenize  from nltk.corpus import stopwords  from collections import defaultdict  from heapq import nlargest  def summarize\_text(text, num\_sentences=3):  """  Generates an extractive summary of a text after preprocessing.  Args:  text (str): The input text paragraph.  num\_sentences (int): The desired number of sentences in the summary.  Returns:  str: The generated extractive summary.  """  # Step 1: Preprocess the text to remove special characters and digits  cleaned\_text = re.sub(r'[^a-zA-Z\s.]', '', text, re.I|re.A)  cleaned\_text = re.sub(r'\s+', ' ', cleaned\_text).strip()  # Handle cases where all text is removed  if not cleaned\_text or cleaned\_text.isspace():  return "Not enough text to generate a summary."  # Tokenize the text into sentences  sentences = sent\_tokenize(cleaned\_text)  if len(sentences) <= num\_sentences:  return cleaned\_text # Return original text if it's already short  # Tokenize the text into words and remove stopwords  stop\_words = set(stopwords.words('english'))  word\_frequencies = defaultdict(int)  for word in word\_tokenize(cleaned\_text):  if word.lower() not in stop\_words and word.isalpha():  word\_frequencies[word.lower()] += 1  # Check if word\_frequencies is empty  if not word\_frequencies:  return "Not enough meaningful words to generate a summary."  # Normalize the word frequencies  max\_frequency = max(word\_frequencies.values())  for word in word\_frequencies.keys():  word\_frequencies[word] = (word\_frequencies[word]/max\_frequency)  # Step 2: Score sentences based on word frequency  sentence\_scores = defaultdict(int)  for sentence in sentences:  for word in word\_tokenize(sentence):  if word.lower() in word\_frequencies:  sentence\_scores[sentence] += word\_frequencies[word.lower()]  # Step 3: Extract the top N sentences for the summary  summary\_sentences = nlargest(num\_sentences, sentence\_scores, key=sentence\_scores.get)  # Join the sentences to form the summary  summary = ' '.join(summary\_sentences)  return summary  # Example usage with a sample paragraph  sample\_paragraph = """  Artificial intelligence (AI) is a rapidly advancing field of computer science.  Its applications range from simple chatbots to complex autonomous vehicles.  AI systems are designed to perform tasks that would normally require human intelligence, such as visual perception, speech recognition, and decision-making.  In recent years, the development of deep learning models has significantly boosted AI's capabilities, leading to major breakthroughs.  The ethical implications of AI, however, are a subject of ongoing debate. Some researchers believe AI could solve some of the world's most complex problems.  While others worry about its potential impact on employment and data privacy.  The future of AI holds great promise and many challenges, with billions of dollars being invested in research and development each year.  """  # Generate a summary with 3 sentences  generated\_summary = summarize\_text(sample\_paragraph, num\_sentences=3)  print("Original Text:")  print(sample\_paragraph)  print("\n--- Extractive Summary ---")  print(generated\_summary)  **2.Consider review message .Perform sentiment analysis on the message .**  import numpy as np  import pandas as pd  #import nltk  import matplotlib.pyplot as plt  airtweets = pd.read\_csv("Tweets.csv")  plsize plt.rcParams["figure.figsize"]  plt.rcParams["figure.figsize"] = plsize  print("Data in CSV file")  print(airtweets.head())  print("Distribution of sentiments across all tweets ")  airtweets.airline\_sentiment.value\_counts().plot(kind='pie', autopct='%1.  0f%%', colors=["Black", "Orange", "green"])  airline\_sentiment = airtweets.groupby(['airline', 'airline\_sentiment']).  airline\_sentiment.count().unstack()  airline\_sentiment.plot(kind='bar',color=['black', 'blue', 'cyan']) |