Perform the following operations using Python on Iris data set.

1. Load the Dataset into pandas data frame.

2. Display information about missing values in the data

3. Display initial statistics.

4. Check the dimensions of the data frame.

5. Display data type of the variable.

6. Apply proper data type conversion

7. Convert categorical variables into quantitative variables using one hot encoding and label encoder

Ans:

```
# Step 1: Load the Dataset into pandas data frame
    ● import pandas as pd
# Assuming the dataset is saved as 'iris.csv' in the current directory
    ● df = pd.read_csv('iris.csv')
# Step 2: Display information about missing values in the data
    ● missing_values = df.isnull().sum()
    ● print("Missing Values:\n", missing_values)
# Step 3: Display initial statistics
    ● initial_statistics = df.describe()
    ● print("Initial Statistics:\n", initial_statistics)
# Step 4: Check the dimensions of the data frame
    ● dimensions = df.shape
    ● print("Dimensions of the DataFrame:", dimensions)
# Step 5: Display data type of the variables
    ● data_types = df.dtypes
    ● print("Data Types of Variables:\n", data_types)
# Step 6: Apply proper data type conversion (if necessary)
        For eg: df['Gender'].replace(['Male', 'Female'], [0, 1], inplace = True)
# No conversion needed for this dataset as all variables seem to have appropriate data types
# Step 7: Convert categorical variables into quantitative variables using one hot encoding and label
encoder
    ● from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# One-hot encoding for categorical variables (if any)
# Assuming there are no categorical variables in this dataset for one-hot encoding

# Label encoding for categorical target variable (if any)
# Assuming the target variable is 'species' and it is categorical
label_encoder = LabelEncoder()
df['species_encoded'] = label_encoder.fit_transform(df['species'])
# Display the modified DataFrame after label encoding
print("Modified DataFrame after Label Encoding:\n", df.head())
```

Perform the following operations using Python by creating student performance dataset.

1. Display Missing Values
2. Replace missing values using any 2 suitable
3. Identify outliers using boxplot and scatterplot
4. Handle outlier using any technique
5. Perform any 2 data normalization technique

Ans:

Step 1: Import necessary libraries

- import pandas as pd
- import numpy as np
- import seaborn as sns
- import matplotlib.pyplot as plt

Step 2:

df = pd.read_csv('Students.csv')

Step 3: Display Missing Values

- missing_values = df.isnull().sum()
- print("Missing Values:\n", missing_values)

Step 4: Replace missing values using mean or median

Replace missing values in 'Age' column with median

- age_median = df['Age'].median()
- df['Age'].fillna(age_median, inplace=True)

Replace missing values in 'Math_Score' column with mean

- math_mean = df['Math_Score'].mean()
- df['Math_Score'].fillna(math_mean, inplace=True)

Step 5: Identify outliers using boxplot and scatterplot

Boxplot

- plt.figure(figsize=(10, 6))
- sns.boxplot(data=df[['Age', 'Math_Score', 'Science_Score']])
- plt.title('Boxplot of Age, Math_Score, and Science_Score')
- plt.show()

Scatterplot

- plt.figure(figsize=(10, 6))
- sns.scatterplot(x='Math_Score', y='Science_Score', data=df)
- plt.title('Scatterplot of Math_Score vs. Science_Score')
- plt.show()

Step 6: Handle outlier using winsorization technique

- from scipy.stats.mstats import winsorize

Apply winsorization to 'Math_Score' and 'Science_Score' columns

- df['Math_Score'] = winsorize(df['Math_Score'], limits=[0.05, 0.05])

- df['Science_Score'] = winsorize(df['Science_Score'], limits=[0.05, 0.05])

Step 7: Perform any 2 data normalization technique
Min-Max Normalization
- df['Math_Score_MinMax'] = (df['Math_Score'] - df['Math_Score'].min()) / (df['Math_Score'].max() - df['Math_Score'].min())
- df['Science_Score_MinMax'] = (df['Science_Score'] - df['Science_Score'].min()) / (df['Science_Score'].max() - df['Science_Score'].min())

Z-score Normalization
- df['Math_Score_ZScore'] = (df['Math_Score'] - df['Math_Score'].mean()) / df['Math_Score'].std()
- df['Science_Score_ZScore'] = (df['Science_Score'] - df['Science_Score'].mean()) / df['Science_Score'].std()

Display the modified DataFrame
- print("Modified DataFrame:\n", df)

Theory:
- Outliers are data points that significantly deviate from the rest of the data.
- Outliers can distort statistical analyses and machine learning models.
- Identifying outliers can be done using visualization techniques like boxplots and scatterplots.
- Common methods for handling outliers include removing them, transforming the data, or using robust statistical methods like winsorization.

Perform the following operations on iris dataset

1. Display mean, median, minimum, maximum, standard deviation
2. Provide mean, median, minimum, maximum, standard deviation for a given dataset by grouping using one of the qualitative (categorical) variable.
3. Display missing values and inconsistencies.
4. Replace missing values using any 2 suitable

Ans:
Step 1: Import necessary libraries
- import pandas as pd
Step 2: Load the Iris dataset
- iris_df = pd.read_csv('iris.csv')
Step 3: Display mean, median, minimum, maximum, standard deviation
- statistics = iris_df.describe()
- print("Descriptive Statistics:\n", statistics)
Step 4: Provide mean, median, minimum, maximum, standard deviation by grouping using one categorical variable
- grouped_stats = iris_df.groupby('species').describe()
- print("Grouped Descriptive Statistics:\n", grouped_stats)
Step 5: Display missing values and inconsistencies
- missing_values = iris_df.isnull().sum()
- print("Missing Values:\n", missing_values)

Step 6: Replace missing values using two suitable methods
Replace missing values with mean

- iris_df_filled_mean = iris_df.fillna(iris_df.mean())

# Replace missing values with median
iris_df_filled_median = iris_df.fillna(iris_df.median())
# Display the modified datasets after replacing missing values

- print("Dataset after replacing missing values with mean:\n", iris_df_filled_mean.head())
- print("Dataset after replacing missing values with median:\n", iris_df_filled_median.head())

Perform the following operation using titanic data set.

1. check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.
2. plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age')
3. Write observations on the inference from the above statistics.

Ans:
Step 1: Import necessary libraries

- import pandas as pd
- import seaborn as sns
- import matplotlib.pyplot as plt

Step 2: Load the Titanic dataset

- titanic_df = pd.read_csv('titanic.csv')

Step 3: Plot a histogram for the distribution of ticket prices (fare)

- plt.figure(figsize=(10, 6))
- sns.histplot(titanic_df['fare'], bins=20, kde=True)
- plt.title('Distribution of Ticket Prices (Fare)')
- plt.xlabel('Fare')
- plt.ylabel('Frequency')
- plt.show()

Step 4: Plot a box plot for distribution of age with respect to each gender and survival status

- plt.figure(figsize=(10, 6))
- sns.boxplot(x='sex', y='age', hue='survived', data=titanic_df)
- plt.title('Distribution of Age with Respect to Gender and Survival Status')
- plt.xlabel('Gender')
- plt.ylabel('Age')
- plt.show()

Perform the following operations on iris dataset

1. List down the features and their types
2. Create a box plot and histogram for each feature in the dataset.

3. Compare distributions and identify outliers.

Ans:
Step 1: Import necessary libraries

- import pandas as pd
- import seaborn as sns
- import matplotlib.pyplot as plt

Step 2: Load the Iris dataset

- iris_df = pd.read_csv('iris.csv')

Step 3: List down the features and their types

- feature_types = iris_df.dtypes
- print("Features and their types:\n", feature_types)

Step 4: Create a box plot and histogram for each feature in the dataset

- for column in iris_df.columns:
- plt.figure(figsize=(10, 6))
- sns.boxplot(x=iris_df[column])
- plt.title(f'Box plot of {column}')
- plt.show()

- plt.figure(figsize=(10, 6))
- sns.histplot(iris_df[column], bins=20, kde=True)
- plt.title(f'Histogram of {column}')
- plt.xlabel(column)
- plt.ylabel('Frequency')
- plt.show()

Certainly! Let's discuss Box Plots and Histograms:

Box Plot:

- A box plot, also known as a box-and-whisker plot, is a graphical representation of the distribution of a dataset.
- It displays the distribution of data based on five summary statistics: minimum, first quartile (Q1), median (second quartile or Q2), third quartile (Q3), and maximum.
- The box represents the interquartile range (IQR), which is the range between the first and third quartiles (Q1 and Q3). The median is shown as a line inside the box.
- The "whiskers" extend from the box to the minimum and maximum values within a certain range (usually 1.5 times the IQR). Data points beyond the whiskers are considered outliers and are plotted individually.
- Box plots are useful for visualizing the spread and central tendency of a dataset, as well as identifying outliers and comparing distributions between different groups or categories.

Histogram:

- A histogram is a graphical representation of the distribution of numerical data.

- It consists of a series of bars, where each bar represents the frequency or count of data points falling within a specific range (bin) of values.
- The x-axis represents the range of values (bins), and the y-axis represents the frequency or count of data points within each bin.
- Histograms provide insights into the shape, spread, and central tendency of a dataset. They allow us to visualize the frequency distribution of data and identify patterns such as peaks, valleys, and gaps.
- Histograms are particularly useful for understanding the distribution of continuous variables and detecting the presence of clusters or outliers within the data.

In summary, both box plots and histograms are powerful tools for visualising and analysing the distribution of data. While box plots focus on summarising the distribution using summary statistics and identifying outliers, histograms provide a more detailed view of the frequency distribution of data within specific ranges.

Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset. Find the performance of your model.

Ans:

```
# Step 1: Import necessary libraries
```

- import pandas as pd
- from sklearn.datasets import load_boston
- from sklearn.model_selection import train_test_split
- from sklearn.linear_model import LinearRegression
- from sklearn.metrics import mean_squared_error, r2_score

```
# Step 2: Load the Boston Housing Dataset
```

- boston = load_boston()
- boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)
- boston_df['PRICE'] = boston.target

```
# Step 3: Split the data into features (X) and target variable (y)
```

- X = boston_df.drop('PRICE', axis=1)
- y = boston_df['PRICE']

```
# Step 4: 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)

```
# Step 5: Create and train the Linear Regression model
```

- model = LinearRegression()
- model.fit(X_train, y_train)

```
# Step 6: Make predictions on the test set
```

- y_pred = model.predict(X_test)

```
# Step 7: Evaluate the performance of the model
```

- mse = mean_squared_error(y_test, y_pred)
- r2 = r2_score(y_test, y_pred)
- print("Mean Squared Error:", mse)
- print("R-squared Score:", r2)

Create a Naïve Bayes classification model using Python on on social network ads.csv dataset. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

Ans:

```
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score

# Step 2: Load the dataset
df = pd.read_csv('Social_Network_Ads.csv')

# Step 3: Preprocess the data (if necessary)
# For simplicity, assume data preprocessing (e.g., handling categorical variables, scaling) has already been done

# Step 4: Split the data into features (X) and target variable (y)
X = df.drop(['User ID', 'Purchased'], axis=1)
y = df['Purchased']

# Step 5: 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)

# Step 6: Create and train the Naïve Bayes classification model
model = GaussianNB()
model.fit(X_train, y_train)

# Step 7: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 8: Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Step 9: Extract TP, FP, TN, FN from confusion matrix
```

```
TP = conf_matrix[1, 1]
FP = conf_matrix[0, 1]
TN = conf_matrix[0, 0]
FN = conf_matrix[1, 0]

# Step 10: Compute evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Step 11: Print the evaluation metrics
print("Confusion Matrix:\n", conf_matrix)
print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
```

Calculate Term Frequency and Inverse Document Frequency. Considering sentences of documents.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Example documents
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?",
]

# Calculate TF-IDF
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)

# Get feature names (terms)
terms = vectorizer.get_feature_names_out()

# Convert TF-IDF matrix to DataFrame for better visualization
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=terms)
print(tfidf_df)
```

Write Scala program to find average temperature, average dew point and average wind speed for given weather dataset

Ans:

```
import scala.io.Source

object WeatherAnalysis {
  def main(args: Array[String]): Unit = {
    // Load the weather dataset from a CSV file
    val weatherDataFile = "weather_dataset.csv"
    val weatherData = Source.fromFile(weatherDataFile).getLines().drop(1) // Skip header line

    // Initialize variables to store total values and count
    var totalTemperature = 0.0
    var totalDewPoint = 0.0
    var totalWindSpeed = 0.0
    var count = 0

    // Iterate over each line in the dataset
    for (line <- weatherData) {
      val columns = line.split(",") // Assuming CSV format

      // Extract temperature, dew point, and wind speed from the columns
      val temperature = columns(0).toDouble
      val dewPoint = columns(1).toDouble
      val windSpeed = columns(2).toDouble

      // Update total values and count
      totalTemperature += temperature
      totalDewPoint += dewPoint
      totalWindSpeed += windSpeed
      count += 1
    }

    // Calculate average temperature, dew point, and wind speed
    val avgTemperature = totalTemperature / count
    val avgDewPoint = totalDewPoint / count
    val avgWindSpeed = totalWindSpeed / count

    // Print the results
    println(s"Average Temperature: $avgTemperature")
    println(s"Average Dew Point: $avgDewPoint")
    println(s"Average Wind Speed: $avgWindSpeed")
  }
```

}

Perform the following operations using Python by creating student performance dataset.

1. Display Missing Values
2. Replace missing values using any 2 suitable
3. Identify outliers using IQR and ZScore
4. Handle outlier using any technique
5. Perform data normalization using Min Max

Ans:

```
import pandas as pd
import numpy as np
from scipy import stats

# Create a sample student performance dataset
data = {
    'Student_ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Math_Score': [80, 85, 90, np.nan, 75, 95, 65, 70, 55, 100],
    'Science_Score': [85, 90, np.nan, 70, 95, 60, 75, 80, 65, 100],
    'English_Score': [90, np.nan, 95, 75, 100, 80, 85, 70, 65, 60]
}

df = pd.DataFrame(data)

# 1. Display Missing Values
print("Missing Values:")
print(df.isnull())

# 2. Replace missing values using Mean and Median
df_filled_mean = df.fillna(df.mean())  # Replace missing values with mean
df_filled_median = df.fillna(df.median())  # Replace missing values with median

# 3. Identify outliers using IQR and ZScore
def detect_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return (data < lower_bound) | (data > upper_bound)
```

```
def detect_outliers_zscore(data, threshold=3):
    z_scores = np.abs(stats.zscore(data))
    return np.where(z_scores > threshold, True, False)

outliers_iqr = detect_outliers_iqr(df_filled_median)
outliers_zscore = detect_outliers_zscore(df_filled_median)

print("\nOutliers using IQR:")
print(outliers_iqr)

print("\nOutliers using ZScore:")
print(outliers_zscore)

# 4. Handle outliers using Winsorization
def winsorize(data):
    lower_bound = data.quantile(0.05)
    upper_bound = data.quantile(0.95)
    data[data < lower_bound] = lower_bound
    data[data > upper_bound] = upper_bound
    return data

df_winsorized = df_filled_median.copy()
for column in df_winsorized.columns[1:]:
    df_winsorized[column] = winsorize(df_winsorized[column])

# 5. Perform data normalization using Min-Max scaling
def min_max_scaling(data):
    return (data - data.min()) / (data.max() - data.min())

df_normalized = df_winsorized.copy()
for column in df_normalized.columns[1:]:
    df_normalized[column] = min_max_scaling(df_normalized[column])

print("\nNormalized Data:")
print(df_normalized)
```

Perform the following operations using Python by creating student performance dataset.

1. Display Missing Values
2. Replace missing values using any 2 suitable
3. Identify outliers using IQR and ZScore

4.Handle outlier using any technique
5.Perform data normalization using decimal scaling
Ans:

```
import pandas as pd
import numpy as np
from scipy import stats

# Create a sample student performance dataset
data = {
    'Student_ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Math_Score': [80, 85, 90, np.nan, 75, 95, 65, 70, 55, 100],
    'Science_Score': [85, 90, np.nan, 70, 95, 60, 75, 80, 65, 100],
    'English_Score': [90, np.nan, 95, 75, 100, 80, 85, 70, 65, 60]
}

df = pd.DataFrame(data)

# 1. Display Missing Values
print("Missing Values:")
print(df.isnull())

# 2. Replace missing values using Mean and Median
df_filled_mean = df.fillna(df.mean())  # Replace missing values with mean
df_filled_median = df.fillna(df.median())  # Replace missing values with median

# 3. Identify outliers using IQR and ZScore
def detect_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return (data < lower_bound) | (data > upper_bound)

def detect_outliers_zscore(data, threshold=3):
    z_scores = np.abs(stats.zscore(data))
    return np.where(z_scores > threshold, True, False)

outliers_iqr = detect_outliers_iqr(df_filled_median)
outliers_zscore = detect_outliers_zscore(df_filled_median)

print("\nOutliers using IQR:")
print(outliers_iqr)
```

```
print("\nOutliers using ZScore:")
print(outliers_zscore)

# 4. Handle outliers using Winsorization
def winsorize(data):
    lower_bound = data.quantile(0.05)
    upper_bound = data.quantile(0.95)
    data[data < lower_bound] = lower_bound
    data[data > upper_bound] = upper_bound
    return data

df_winsorized = df_filled_median.copy()
for column in df_winsorized.columns[1:]:
    df_winsorized[column] = winsorize(df_winsorized[column])

# 5. Perform data normalization using Decimal Scaling
def decimal_scaling(data):
    max_value = data.abs().max()
    num_digits = len(str(int(max_value)))
    scaled_data = data / (10 ** num_digits)
    return scaled_data

df_normalized = df_winsorized.copy()
for column in df_normalized.columns[1:]:
    df_normalized[column] = decimal_scaling(df_normalized[column])

print("\nNormalized Data:")
print(df_normalized)
```

For given text apply following preprocessing methods:
1. Tokenization
2. POS Tagging
3. Stop word Removal

Ans:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
```

```
# Sample text
text = "Natural Language Processing is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data."

# 1. Tokenization
tokens = word_tokenize(text)

# 2. POS Tagging
pos_tags = pos_tag(tokens)

# 3. Stop word Removal
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

print("Tokens:")
print(tokens)
print("\nPOS Tags:")
print(pos_tags)
print("\nFiltered Tokens after Stop word Removal:")
print(filtered_tokens)
```