**1)# Program to describe iris dataset using pandas dataframe commands and encode the same**

'''

Source for theory

https://jay190301.medium.com/data-science-data-pre-processing-using-scikit-learn-iris-dataset-1ba0a9ae04e6

'''

# Ceated on 21-01-2024 at 11:00am

from sklearn import datasets

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Import the dataset

# From csv file

## data = pd.read\_csv("D:/Input/ML\_lab/iris\_csv.csv")

# From scikit-learn datasets

iris = datasets.load\_iris()

# Convert the dataset into pandas dataframe

data = pd.DataFrame(data= np.c\_[iris['data'], iris['target']], columns= iris['feature\_names'] + ['target'])

# Add the target names or label column to the dataframe

data['target\_names'] = data['target'].replace(dict(enumerate(iris.target\_names)))

# Printing top 5 rows

# print(data.head())

'''

Output:

sepal length (cm) sepal width (cm) ... target target\_names

0 5.1 3.5 ... 0.0 setosa

1 4.9 3.0 ... 0.0 setosa

2 4.7 3.2 ... 0.0 setosa

3 4.6 3.1 ... 0.0 setosa

4 5.0 3.6 ... 0.0 setosa

[5 rows x 6 columns]

'''

# Display the information iris dataset

# data.info()

"""

Output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 sepal length (cm) 150 non-null float64

1 sepal width (cm) 150 non-null float64

2 petal length (cm) 150 non-null float64

3 petal width (cm) 150 non-null float64

4 target 150 non-null float64

5 target\_names 150 non-null object

dtypes: float64(5), object(1)

memory usage: 7.2+ KB

"""

# Describe the iris dataset

# print(data.describe())

'''

Output:

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

count 150.000000 150.000000 150.000000 150.000000 150.000000

mean 5.843333 3.057333 3.758000 1.199333 1.000000

std 0.828066 0.435866 1.765298 0.762238 0.819232

min 4.300000 2.000000 1.000000 0.100000 0.000000

25% 5.100000 2.800000 1.600000 0.300000 0.000000

50% 5.800000 3.000000 4.350000 1.300000 1.000000

75% 6.400000 3.300000 5.100000 1.800000 2.000000

max 7.900000 4.400000 6.900000 2.500000 2.000000

'''

# Encoding process

# Type I - Label encodinng

# Encoder function

Label\_encoder = LabelEncoder()

# Labels

data['target\_names'] = Label\_encoder.fit\_transform(data['target\_names'])

# print(data['target\_names'])

'''

Output

0 0

1 0

2 0

3 0

4 0

..

145 2

146 2

147 2

148 2

149 2

Name: target\_names, Length: 150, dtype: int32

'''

# Actual labels

# print(Label\_encoder.classes\_)

'''

Output

'setosa' 'versicolor' 'virginica']

'''

# Label count after encoding

# print(data['target\_names'].value\_counts())

'''

0 50

1 50

2 50

Name: target\_names, dtype: int64

'''

# Type II - Label encodinng

# Encoder function

One\_hot = OneHotEncoder()

# Data transformation and conversion to an array

transformed\_data = One\_hot.fit\_transform(data['target\_names'].values.reshape(-1,1)).toarray()

print(One\_hot.categories\_)

'''

Output

[array([0., 1., 2.])]

'''

# Encode all the labels in the dataset

transformed\_data = pd.DataFrame(transformed\_data, columns=['setosa','versicolor','virginica'])

print(transformed\_data.head())

'''

Output

setosa versicolor virginica

0 1.0 0.0 0.0

1 1.0 0.0 0.0

2 1.0 0.0 0.0

3 1.0 0.0 0.0

4 1.0 0.0 0.0

'''

**2)# Program to normalize the iris dataset using standa**

'''

Source for theory

https://jay190301.medium.com/data-science-data-pre-processing-using-scikit-learn-iris-dataset-1ba0a9ae04e6

'''

# Ceated on 21-01-2024 at 11:30am

from sklearn import datasets

import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler

import warnings

# Import the dataset

iris = datasets.load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

print(data)

print('Average of Feature')

print(data.mean())

'''

Output:

Average of Feature

sepal length (cm) 5.843333

sepal width (cm) 3.057333

petal length (cm) 3.758000

petal width (cm) 1.199333

dtype: float64

'''

print('\nFeature Variance')

print(data.var())

'''

Output:

Feature Variance

sepal length (cm) 0.685694

sepal width (cm) 0.189979

petal length (cm) 3.116278

petal width (cm) 0.581006

dtype: float64

'''

# Standardize every features

# create StandardScaler() object

scaler = StandardScaler()

# fit() & transform()

scaler.fit(data)

iris\_scaled\_std = scaler.transform(data)

# transform() returns ndarray -> convert to dataframe

iris\_df\_std\_scaled = pd.DataFrame(data=iris\_scaled\_std, columns=iris.feature\_names)

print('Standardized Feature Average')

print(iris\_df\_std\_scaled.mean())

'''

Output:

Standardized Feature Average

sepal length (cm) -1.690315e-15

sepal width (cm) -1.842970e-15

petal length (cm) -1.698641e-15

petal width (cm) -1.409243e-15

dtype: float64

'''

print('\nStandardized Feature Variance')

print(iris\_df\_std\_scaled.var())

'''

Output:

Standardized Feature Variance

sepal length (cm) 1.006711

sepal width (cm) 1.006711

petal length (cm) 1.006711

petal width (cm) 1.006711

'''

# create MinMaxScaler() object

scaler = MinMaxScaler()

# fit() & transform()

scaler.fit(data)

iris\_scaled\_minmax = scaler.transform(data)

# transform() returns ndarray -> convert to dataframe

iris\_df\_minmax\_scaled = pd.DataFrame(data=iris\_scaled\_minmax, columns=iris.feature\_names)

print('MinMax Scaled Feature Average')

print(iris\_df\_minmax\_scaled.mean())

'''

Output:

MinMax Scaled Feature Average

sepal length (cm) 0.428704

sepal width (cm) 0.440556

petal length (cm) 0.467458

petal width (cm) 0.458056

dtype: float64

'''

print('\nMinMax Scaled Feature Variance')

print(iris\_df\_minmax\_scaled.var())

'''

Output:

MinMax Scaled Feature Variance

sepal length (cm) 0.052908

sepal width (cm) 0.032983

petal length (cm) 0.089522

petal width (cm) 0.100869

dtype: float64

'''

print('\nMinMax Scaled Min Value')

print(iris\_df\_minmax\_scaled.min())

'''

Output:

MinMax Scaled Min Value

sepal length (cm) 0.0

sepal width (cm) 0.0

petal length (cm) 0.0

petal width (cm) 0.0

dtype: float64

'''

print('\nMinMax Scaled Max Value')

print(iris\_df\_minmax\_scaled.max())

'''

Output:

MinMax Scaled Max Value

sepal length (cm) 1.0

sepal width (cm) 1.0

petal length (cm) 1.0

petal width (cm) 1.0

dtype: float64

'''

3) # Program to impute the missing values and perfom discrete transformation on the iris dataset using

'''

Source for theory

https://jay190301.medium.com/data-science-data-pre-processing-using-scikit-learn-iris-dataset-1ba0a9ae04e6

'''

# Ceated on 21-01-2024 at 11:30am

from sklearn import datasets

import numpy as np

import pandas as pd

from sklearn.preprocessing import KBinsDiscretizer

from sklearn.impute import SimpleImputer

import warnings

# Import the dataset

iris = datasets.load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

# print(data)

'''

Output:

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

0 5.1 3.5 1.4 0.2

1 4.9 3.0 1.4 0.2

2 4.7 3.2 1.3 0.2

3 4.6 3.1 1.5 0.2

4 5.0 3.6 1.4 0.2

.. ... ... ... ...

145 6.7 3.0 5.2 2.3

146 6.3 2.5 5.0 1.9

147 6.5 3.0 5.2 2.0

148 6.2 3.4 5.4 2.3

149 5.9 3.0 5.1 1.8

'''

# Define the imputer

imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')

# print(imputer)

# Fit the dataset and transform

imputer = imputer.fit(data)

imputed\_dataset = imputer.transform(data)

# Display the dataframe and check whether missing values are there or not

# print(imputed\_dataset)

'''

Output:

[[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

[5. 3.4 1.5 0.2]

.....]

'''

# Quantile Discretization Transform

# Define the transform function

iris\_quantile\_transform = KBinsDiscretizer(n\_bins=10, encode='ordinal', strategy='quantile')

# Fit the dataset

quantile\_transformed\_data = iris\_quantile\_transform.fit\_transform(data)

# print(pd.DataFrame(quantile\_transformed\_data))

'''

Output:

0 1 2 3

0 2.0 7.0 1.0 1.0

1 1.0 4.0 1.0 1.0

2 0.0 6.0 0.0 1.0

3 0.0 5.0 2.0 1.0

4 2.0 7.0 1.0 1.0

.. ... ... ... ...

145 8.0 4.0 7.0 8.0

146 7.0 1.0 7.0 7.0

147 7.0 4.0 7.0 7.0

148 6.0 7.0 8.0 8.0

149 5.0 4.0 7.0 6.0

[150 rows x 4 columns]

'''

# Uniform Discretization Transform

# Discretization - Discretization is the process through which we can transform continuous variables, models or functions into a discrete form

# Define the transform function

iris\_uniform\_transform = KBinsDiscretizer(n\_bins=10, encode='ordinal', strategy='uniform')

# Fit the dataset

uniform\_transformed\_data = iris\_uniform\_transform.fit\_transform(data)

print(pd.DataFrame(uniform\_transformed\_data))

'''

Output:

0 1 2 3

0 2.0 6.0 0.0 0.0

1 1.0 4.0 0.0 0.0

2 1.0 5.0 0.0 0.0

3 0.0 4.0 0.0 0.0

4 1.0 6.0 0.0 0.0

.. ... ... ... ...

145 6.0 4.0 7.0 9.0

146 5.0 2.0 6.0 7.0

147 6.0 4.0 7.0 7.0

148 5.0 5.0 7.0 9.0

149 4.0 4.0 6.0 7.0

[150 rows x 4 columns]

'''

**4)# Program to visualize the iris dataset using sepal and petal values**

**'''**

Source for theory

https://scikit-learn.org/stable/auto\_examples/datasets/plot\_iris\_dataset.html#sphx-glr-auto-examples-datasets-plot-iris-dataset-py

'''

# Ceated on 21-01-2024 at 11:30am

from sklearn import datasets

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Import the dataset

iris = datasets.load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

# print(data)

'''

Output:

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

0 5.1 3.5 1.4 0.2

1 4.9 3.0 1.4 0.2

2 4.7 3.2 1.3 0.2

3 4.6 3.1 1.5 0.2

4 5.0 3.6 1.4 0.2

.. ... ... ... ...

145 6.7 3.0 5.2 2.3

146 6.3 2.5 5.0 1.9

147 6.5 3.0 5.2 2.0

148 6.2 3.4 5.4 2.3

149 5.9 3.0 5.1 1.8

'''

# Based on sepal length (cm) and sepal width (cm)

# Define the plot

\_, ax = plt.subplots()

# Define the scatter plot for iris dataset

scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)

# Fit the iris dataset to plot's axis

ax.set(xlabel=iris.feature\_names[0], ylabel=iris.feature\_names[1])

# Draw the plot

\_ = ax.legend(scatter.legend\_elements()[0], iris.target\_names, loc="lower right", title="Classes")

plt.show()

# Based on petal length (cm) and petal width (cm)

# Define the plot

\_, ax = plt.subplots()

# Define the scatter plot for iris dataset

scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)

# Fit the iris dataset to plot's axis

ax.set(xlabel=iris.feature\_names[2], ylabel=iris.feature\_names[3])

# Draw the plot

\_ = ax.legend(scatter.legend\_elements()[0], iris.target\_names, loc="lower right", title="Classes")

plt.show()

**5) Program to apply Principal Component Analysis on the iris dataset**

**'''**

Source for theory

https://scikit-learn.org/stable/auto\_examples/decomposition/plot\_pca\_iris.html#sphx-glr-auto-examples-decomposition-plot-pca-iris-py

'''

# Ceated on 21-01-2024 at 11:30am

from sklearn import datasets, decomposition

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import mpl\_toolkits.mplot3d

# Import the dataset

iris = datasets.load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

X = data

y = iris.target

# print(data)

'''

Output:

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

0 5.1 3.5 1.4 0.2

1 4.9 3.0 1.4 0.2

2 4.7 3.2 1.3 0.2

3 4.6 3.1 1.5 0.2

4 5.0 3.6 1.4 0.2

.. ... ... ... ...

145 6.7 3.0 5.2 2.3

146 6.3 2.5 5.0 1.9

147 6.5 3.0 5.2 2.0

148 6.2 3.4 5.4 2.3

149 5.9 3.0 5.1 1.8

'''

# Define the plot

fig = plt.figure(1, figsize=(4, 3))

plt.clf()

ax = fig.add\_subplot(111, projection="3d", elev=48, azim=134)

ax.set\_position([0, 0, 0.95, 1])

# Apply PCA to dataset and get the new data componants

plt.cla()

pca = decomposition.PCA(n\_components=3)

pca.fit(X)

X = pca.transform(X)

# print(X)

for name, label in [("Setosa", 0), ("Versicolour", 1), ("Virginica", 2)]:

ax.text3D(

X[y == label, 0].mean(),

X[y == label, 1].mean() + 1.5,

X[y == label, 2].mean(),

name,

horizontalalignment="center",

bbox=dict(alpha=0.5, edgecolor="w", facecolor="w"),

)

# Reorder the labels to have colors matching the cluster results

y = np.choose(y, [1, 2, 0]).astype(float)

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap=plt.cm.nipy\_spectral, edgecolor="k")

ax.xaxis.set\_ticklabels([])

ax.yaxis.set\_ticklabels([])

ax.zaxis.set\_ticklabels([])

plt.show()