pratical-1

April 17, 2024

1 Data Wrangling - I

Perform the following operations using Python on any open source dataset (e.g., data.csv) 1. Import all the required Python Libraries. 2. Locate open source data from the web (e.g., https://www.kaggle.com). Provide a clear description of the data and its source (i.e., URL of the web site). 3. Load the Dataset into pandas dataframe. 4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame. 5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions. 6. Turn categorical variables into quantitative variables in Python. In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set.

[ ]:

```
# Import the required libraries
# import pandas as pd
# import numpy as np
# import sklearn
# from sklearn import datasets
# import matplotlib.pyplot as plt
# import seaborn as sns
# from IPython.display import display

# csv_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/
 ↪iris.data'
# # Load iris.csv file into a Pandas Data_Frame
# iris = pd.read_csv(csv_url,header = None)
# iris

# # The csv file at the UCI repository does not contain the variable/column␣
 ↪names
# col_names =␣
 ↪['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width','Species']
# col_names
# iris = pd.read_csv(csv_url, names = col_names)
# iris

# # # Data Preprocessing
# # Display First 5 rows
# iris.head()

# # Display Last 5 rows
# iris.tail()

# # The index (row labels) of the Dataset
# iris.index

# # The column labels of the Dataset
# iris.columns

# # Return a tuple representing the dimensionality of datatset
# iris.shape

# # Return the dtypes in the Dataset
# iris.dtypes

# # Return the columns values in the Dataset in array format
# iris.columns.values

# # Generate descriptive statistics to view some basic statistical details
# iris.describe(include='all')

# # Read the Data Column wise
# iris[['Petal_Length','Petal_Width']]

# # Sort object by labels (along an axis)
# iris.sort_index(axis=1, ascending=False)

# # Sort values by column name
# iris.sort_values(by="Petal_Width")

# # # Few Examples of iLoc to slice data for iris Dataset

# # Purely integer-location based indexing for selection by position
# iris.iloc[5]

# # Selecting via [], which slices the rows
# iris.iloc[0:3]

# # Slice the data
# iris.iloc[3:6,0:5]
# iris['Sepal_Length'].iloc[5]

# col_1_3 = iris.columns[1:3]
# iris[col_1_3]

# # In one Expression answer for the above commands
# iris[iris.columns[1:3]].iloc[5:10]

# # Checking of Missing Values

# iris.isnull()

# iris.isnull().any()

# iris.isnull().sum()

# iris.isnull().sum().sum()

# iris.isnull().sum(axis = 1)

# iris.Sepal_Length.isnull()

# iris.Sepal_Length.isnull()

# function = lambda x: x.isnull().sum()
# iris.groupby(['Sepal_Length'])['Petal_Length'].apply(function)

# # Data Formatting and Normalization

# # 1) Data Formatting
# df = iris

# df.dtypes

# # To change the data type
# df['Petal_Length']= df['Petal_Length'].astype("int")

# df.dtypes

# # 2) Data Normalization
# from sklearn import preprocessing

# # Load the iris dataset in dataframe object df
# iris_1 = datasets.load_iris()
# df1 = pd.DataFrame(iris_1.data, columns = iris_1.feature_names)
# # Print df1
# df1

# df1.head()

# # Algorithm for normalization

# # Create x, where x the column's values as floats
# x = df1[['sepal length (cm)','sepal width (cm)', 'petal length (cm)', 'petal␣
 ↪width (cm)']].values.astype(float)

# # Create a minimum and maximum processor object
# min_max_scaler = preprocessing.MinMaxScaler()

# # Create an object to transform the data to fit minmax processor
# x_scaled = min_max_scaler.fit_transform(x)

# # Run the normalizer on the dataframe
# df1_normalized = pd.DataFrame(x_scaled)

# # View the data frame
# df1_normalized

# # Turn categorical variables into quantitative variables in Python

# # 1) Label Encoding
# # Algorithm for label encoding

# # Observe the unique values for the Species column
# df['Species'].unique()

# # define label_encoder object knows how to understand word labels
# label_encoder = preprocessing.LabelEncoder()

# # Encode labels in column 'species'
# df['Species']= label_encoder.fit_transform(df['Species'])

# # Observe the unique values for the Species column
# df['Species'].unique()
```

[1]:

```
# Import the required libraries
import pandas as pd
import numpy as np
import sklearn
from sklearn import datasets
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from IPython.display import display
```

[2]:

```
csv_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.
  ↪data'
# Load iris.csv file into a Pandas Data_Frame
iris = pd.read_csv(csv_url,header = None)
iris
```

[2]:

```
       0    1    2    3               4
0    5.1  3.5  1.4  0.2     Iris-setosa
1    4.9  3.0  1.4  0.2     Iris-setosa
2    4.7  3.2  1.3  0.2     Iris-setosa
3    4.6  3.1  1.5  0.2     Iris-setosa
4    5.0  3.6  1.4  0.2     Iris-setosa
..   …    …    …    …               …
145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica

[150 rows x 5 columns]
```

[3]:

```
# The csv file at the UCI repository does not contain the variable/column names
col_names =␣
  ↪['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width','Species']
```

[4]:

```
col_names
```

[4]:

```
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width', 'Species']
```

[5]:

```
iris = pd.read_csv(csv_url, names = col_names)
```

[6]:

```
iris
```

[6]:

```
     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width         Species
0             5.1          3.5           1.4          0.2     Iris-setosa
1             4.9          3.0           1.4          0.2     Iris-setosa
2             4.7          3.2           1.3          0.2     Iris-setosa
3             4.6          3.1           1.5          0.2     Iris-setosa
4             5.0          3.6           1.4          0.2     Iris-setosa
..            …            …             …            …               …
145           6.7          3.0           5.2          2.3  Iris-virginica
146           6.3          2.5           5.0          1.9  Iris-virginica
147           6.5          3.0           5.2          2.0  Iris-virginica
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica
```

```
[150 rows x 5 columns]
```

1.1 Data Preprocessing

```
[7]: # Display First 5 rows
     iris.head()
```

```
[7]:    Sepal_Length  Sepal_Width  Petal_Length  Petal_Width      Species
     0           5.1          3.5           1.4          0.2  Iris-setosa
     1           4.9          3.0           1.4          0.2  Iris-setosa
     2           4.7          3.2           1.3          0.2  Iris-setosa
     3           4.6          3.1           1.5          0.2  Iris-setosa
     4           5.0          3.6           1.4          0.2  Iris-setosa
```

```
[8]: # Display Last 5 rows
     iris.tail()
```

```
[8]:      Sepal_Length  Sepal_Width  Petal_Length  Petal_Width         Species
     145           6.7          3.0           5.2          2.3  Iris-virginica
     146           6.3          2.5           5.0          1.9  Iris-virginica
     147           6.5          3.0           5.2          2.0  Iris-virginica
     148           6.2          3.4           5.4          2.3  Iris-virginica
     149           5.9          3.0           5.1          1.8  Iris-virginica
```

```
[9]: # The index (row labels) of the Dataset
     iris.index
```

```
[9]: RangeIndex(start=0, stop=150, step=1)
```

```
[10]: # The column labels of the Dataset
      iris.columns
```

```
[10]: Index(['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width',
             'Species'],
            dtype='object')
```

```
[11]: # Return a tuple representing the dimensionality of datatset
      iris.shape
```

```
[11]: (150, 5)
```

```
[12]: # Return the dtypes in the Dataset
      iris.dtypes
```

```
[12]: Sepal_Length    float64
      Sepal_Width     float64
```

```
Petal_Length    float64
Petal_Width     float64
Species          object
dtype: object
```

```
[13]: # Return the columns values in the Dataset in array format
      iris.columns.values
```

```
[13]: array(['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width',
             'Species'], dtype=object)
```

```
[14]: # Generate descriptive statistics to view some basic statistical details
      iris.describe(include='all')
```

```
[14]:         Sepal_Length  Sepal_Width  Petal_Length  Petal_Width      Species
      count     150.000000   150.000000    150.000000   150.000000          150
      unique           NaN          NaN           NaN          NaN            3
      top              NaN          NaN           NaN          NaN  Iris-setosa
      freq             NaN          NaN           NaN          NaN           50
      mean        5.843333     3.054000      3.758667     1.198667          NaN
      std         0.828066     0.433594      1.764420     0.763161          NaN
      min         4.300000     2.000000      1.000000     0.100000          NaN
      25%         5.100000     2.800000      1.600000     0.300000          NaN
      50%         5.800000     3.000000      4.350000     1.300000          NaN
      75%         6.400000     3.300000      5.100000     1.800000          NaN
      max         7.900000     4.400000      6.900000     2.500000          NaN
```

```
[15]: # Read the Data Column wise
      iris[['Petal_Length','Petal_Width']]
```

```
[15]:      Petal_Length  Petal_Width
      0             1.4          0.2
      1             1.4          0.2
      2             1.3          0.2
      3             1.5          0.2
      4             1.4          0.2
      ..            …            …
      145           5.2          2.3
      146           5.0          1.9
      147           5.2          2.0
      148           5.4          2.3
      149           5.1          1.8

      [150 rows x 2 columns]
```

```
[16]: # Sort object by labels (along an axis)
      iris.sort_index(axis=1, ascending=False)
```

[16]:
```
             Species  Sepal_Width  Sepal_Length  Petal_Width  Petal_Length
0        Iris-setosa          3.5           5.1          0.2           1.4
1        Iris-setosa          3.0           4.9          0.2           1.4
2        Iris-setosa          3.2           4.7          0.2           1.3
3        Iris-setosa          3.1           4.6          0.2           1.5
4        Iris-setosa          3.6           5.0          0.2           1.4
..               …            …             …            …             …
145   Iris-virginica          3.0           6.7          2.3           5.2
146   Iris-virginica          2.5           6.3          1.9           5.0
147   Iris-virginica          3.0           6.5          2.0           5.2
148   Iris-virginica          3.4           6.2          2.3           5.4
149   Iris-virginica          3.0           5.9          1.8           5.1

[150 rows x 5 columns]
```

[17]:
```python
# Sort values by column name
iris.sort_values(by="Petal_Width")
```

[17]:
```
     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width         Species
32            5.2          4.1           1.5          0.1     Iris-setosa
13            4.3          3.0           1.1          0.1     Iris-setosa
37            4.9          3.1           1.5          0.1     Iris-setosa
9             4.9          3.1           1.5          0.1     Iris-setosa
12            4.8          3.0           1.4          0.1     Iris-setosa
..              …            …             …            …               …
140           6.7          3.1           5.6          2.4  Iris-virginica
114           5.8          2.8           5.1          2.4  Iris-virginica
100           6.3          3.3           6.0          2.5  Iris-virginica
144           6.7          3.3           5.7          2.5  Iris-virginica
109           7.2          3.6           6.1          2.5  Iris-virginica

[150 rows x 5 columns]
```

1.1.1 *Few Examples of iLoc to slice data for iris Dataset*

[18]:
```python
# Purely integer-location based indexing for selection by position
iris.iloc[5]
```

[18]:
```
Sepal_Length            5.4
Sepal_Width             3.9
Petal_Length            1.7
Petal_Width             0.4
Species         Iris-setosa
Name: 5, dtype: object
```

[19]:
```python
# Selecting via [], which slices the rows
iris.iloc[0:3]
```

[19]:

```
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width      Species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
```

[20]:

```
# Slice the data
iris.iloc[3:6,0:5]
```

[20]:

```
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width      Species
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
5           5.4          3.9           1.7          0.4  Iris-setosa
```

[21]:

```
iris['Sepal_Length'].iloc[5]
```

[21]:

```
5.4
```

[22]:

```
col_1_3 = iris.columns[1:3]
iris[col_1_3]
```

[22]:

```
     Sepal_Width  Petal_Length
0            3.5           1.4
1            3.0           1.4
2            3.2           1.3
3            3.1           1.5
4            3.6           1.4
..           …            …
145          3.0           5.2
146          2.5           5.0
147          3.0           5.2
148          3.4           5.4
149          3.0           5.1

[150 rows x 2 columns]
```

[23]:

```
# In one Expression answer for the above commands
iris[iris.columns[1:3]].iloc[5:10]
```

[23]:

```
   Sepal_Width  Petal_Length
5          3.9           1.7
6          3.4           1.4
7          3.4           1.5
8          2.9           1.4
9          3.1           1.5
```

1.1.2 *Checking of Missing Values*

```
[24]: iris.isnull()
```

```
[24]:      Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
      0           False        False         False        False    False
      1           False        False         False        False    False
      2           False        False         False        False    False
      3           False        False         False        False    False
      4           False        False         False        False    False
      ..            …          …           …          …      …
      145         False        False         False        False    False
      146         False        False         False        False    False
      147         False        False         False        False    False
      148         False        False         False        False    False
      149         False        False         False        False    False

      [150 rows x 5 columns]
```

```
[25]: iris.isnull().any()
```

```
[25]: Sepal_Length    False
      Sepal_Width     False
      Petal_Length    False
      Petal_Width     False
      Species         False
      dtype: bool
```

```
[26]: iris.isnull().sum()
```

```
[26]: Sepal_Length    0
      Sepal_Width     0
      Petal_Length    0
      Petal_Width     0
      Species         0
      dtype: int64
```

```
[27]: iris.isnull().sum().sum()
```

```
[27]: 0
```

```
[28]: iris.isnull().sum(axis = 1)
```

```
[28]: 0      0
      1      0
      2      0
      3      0
```

```
4      0
      ..
145    0
146    0
147    0
148    0
149    0
Length: 150, dtype: int64
```

[29]:
```
iris.Sepal_Length.isnull()
```

[29]:
```
0      False
1      False
2      False
3      False
4      False
       …
145    False
146    False
147    False
148    False
149    False
Name: Sepal_Length, Length: 150, dtype: bool
```

[30]:
```
iris.Sepal_Length.isnull().sum()c
```

[30]:
```
0
```

[31]:
```
function = lambda x: x.isnull().sum()
iris.groupby(['Sepal_Length'])['Petal_Length'].apply(function)
```

[31]:
```
Sepal_Length
4.3    0
4.4    0
4.5    0
4.6    0
4.7    0
4.8    0
4.9    0
5.0    0
5.1    0
5.2    0
5.3    0
5.4    0
5.5    0
5.6    0
5.7    0
```

```
5.8    0
5.9    0
6.0    0
6.1    0
6.2    0
6.3    0
6.4    0
6.5    0
6.6    0
6.7    0
6.8    0
6.9    0
7.0    0
7.1    0
7.2    0
7.3    0
7.4    0
7.6    0
7.7    0
7.9    0
Name: Petal_Length, dtype: int64
```

1.2 Data Formatting and Normalization

1.2.1 1) Data Formatting

```
[32]: df = iris
```

```
[33]: df.dtypes
```

```
[33]: Sepal_Length    float64
      Sepal_Width     float64
      Petal_Length    float64
      Petal_Width     float64
      Species          object
      dtype: object
```

```
[34]: # To change the data type
      df['Petal_Length']= df['Petal_Length'].astype("int")
```

```
[35]: df.dtypes
```

```
[35]: Sepal_Length    float64
      Sepal_Width     float64
      Petal_Length      int32
      Petal_Width     float64
      Species          object
```

```
dtype: object
```

1.2.2 2) Data Normalization

[36]:

```
from sklearn import preprocessing
```

[37]:

```
# Load the iris dataset in dataframe object df
iris_1 = datasets.load_iris()
df1 = pd.DataFrame(iris_1.data, columns = iris_1.feature_names)

# Print df1
df1
```

[37]:

```
     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

[150 rows x 4 columns]
```

[38]:

```
df1.head()
```

[38]:

```
   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
```

Algorithm for normalization

[39]:

```
# Create x, where x the column's values as floats

x = df1[['sepal length (cm)','sepal width (cm)', 'petal length (cm)', 'petal␣
 ↪width (cm)']].values.astype(float)
```

[40]:

```
# Create a minimum and maximum processor object
min_max_scaler = preprocessing.MinMaxScaler()
```

[41]:

```
# Create an object to transform the data to fit minmax processor
x_scaled = min_max_scaler.fit_transform(x)
```

[42]:

```
# Run the normalizer on the dataframe
df1_normalized = pd.DataFrame(x_scaled)
```

[43]:

```
# View the data frame
df1_normalized
```

[43]:

```
            0         1         2         3
0    0.222222  0.625000  0.067797  0.041667
1    0.166667  0.416667  0.067797  0.041667
2    0.111111  0.500000  0.050847  0.041667
3    0.083333  0.458333  0.084746  0.041667
4    0.194444  0.666667  0.067797  0.041667
..        …         …         …         …
145  0.666667  0.416667  0.711864  0.916667
146  0.555556  0.208333  0.677966  0.750000
147  0.611111  0.416667  0.711864  0.791667
148  0.527778  0.583333  0.745763  0.916667
149  0.444444  0.416667  0.694915  0.708333

[150 rows x 4 columns]
```

1.3 Turn categorical variables into quantitative variables in Python

1.3.1 1) Label Encoding

Algorithm for label encoding

[44]:

```
# Observe the unique values for the Species column
df['Species'].unique()
```

[44]:

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

[45]:

```
# define label_encoder object knows how to understand word labels
label_encoder = preprocessing.LabelEncoder()
```

[46]:

```
# Encode labels in column 'species'
df['Species']= label_encoder.fit_transform(df['Species'])
```

[47]:

```
# Observe the unique values for the Species column
df['Species'].unique()
```

[47]:

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

1.3.2 2) One Hot Encoding

Algorithm for one hot encoding

[48]:
```
# Observe the unique values for the Species column
unique_species = df['Species'].unique()
print("Unique species:", unique_species)
```

```
Unique species: [0 1 2]
```

[49]:
```
# define label_encoder object knows how to understand word labels
label_encoder = preprocessing.LabelEncoder()
df['Species_encoded'] = label_encoder.fit_transform(df['Species'])
```

[50]:
```
# Remove the target variable from dataset
df_features = df.drop(columns=['Species', 'Species_encoded'])
```

[51]:
```
# Apply one_hot encoder for Species column
one_hot_encoder = preprocessing.OneHotEncoder()
species_encoded_one_hot = one_hot_encoder.
 ↪fit_transform(df[['Species_encoded']]).toarray()
```

[52]:
```
# Join the encoded values with Features variable
df_encoded = pd.concat([df_features, pd.DataFrame(species_encoded_one_hot)],␣
 ↪axis=1)
```

[53]:
```
# Observe the merge dataframe
print("\nMerged DataFrame:")
display(df_encoded)
```

```
Merged DataFrame:

     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    0    1    2
0             5.1          3.5             1          0.2  1.0  0.0  0.0
1             4.9          3.0             1          0.2  1.0  0.0  0.0
2             4.7          3.2             1          0.2  1.0  0.0  0.0
3             4.6          3.1             1          0.2  1.0  0.0  0.0
4             5.0          3.6             1          0.2  1.0  0.0  0.0
..            …            …             …            …   …    …   …
145           6.7          3.0             5          2.3  0.0  0.0  1.0
146           6.3          2.5             5          1.9  0.0  0.0  1.0
147           6.5          3.0             5          2.0  0.0  0.0  1.0
148           6.2          3.4             5          2.3  0.0  0.0  1.0
149           5.9          3.0             5          1.8  0.0  0.0  1.0

[150 rows x 7 columns]
```

[54]:
```
# Rename the newly encoded columns
```

```
encoded_column_names = {0: 'Iris-Setosa', 1: 'Iris-Versicolor', 2:␣
 ↪'Iris-virginica'}
df_encoded.rename(columns=encoded_column_names, inplace=True)
```

[55]:

```
# Observing the merged DataFrame with renamed columns
print("Merged DataFrame with renamed columns:")
display(df_encoded)
```

```
Merged DataFrame with renamed columns:

     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Iris-Setosa  \
0             5.1          3.5             1          0.2          1.0
1             4.9          3.0             1          0.2          1.0
2             4.7          3.2             1          0.2          1.0
3             4.6          3.1             1          0.2          1.0
4             5.0          3.6             1          0.2          1.0
..            …          …           …          …          …
145           6.7          3.0             5          2.3          0.0
146           6.3          2.5             5          1.9          0.0
147           6.5          3.0             5          2.0          0.0
148           6.2          3.4             5          2.3          0.0
149           5.9          3.0             5          1.8          0.0

     Iris-Versicolor  Iris-virginica
0                0.0             0.0
1                0.0             0.0
2                0.0             0.0
3                0.0             0.0
4                0.0             0.0
..               …            …
145              0.0             1.0
146              0.0             1.0
147              0.0             1.0
148              0.0             1.0
149              0.0             1.0

[150 rows x 7 columns]
```

1.3.3 3) Dummy Variable Encoding

Algorithm

[56]:

```
df['Species'].unique()
```

[56]:

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

[57]:

```
label_encoder = preprocessing.LabelEncoder()
df['Species_encoded'] = label_encoder.fit_transform(df['Species'])
```

[58]:

```
one_hot_df = pd.get_dummies(df, prefix="Species",columns=['Species'],␣
 ↪drop_first=False).astype(float)
```

[59]:

```
one_hot_df
```

[59]:

```
     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species_encoded  \
0             5.1          3.5           1.0          0.2              0.0
1             4.9          3.0           1.0          0.2              0.0
2             4.7          3.2           1.0          0.2              0.0
3             4.6          3.1           1.0          0.2              0.0
4             5.0          3.6           1.0          0.2              0.0
..            …            …             …            …               …
145           6.7          3.0           5.0          2.3              2.0
146           6.3          2.5           5.0          1.9              2.0
147           6.5          3.0           5.0          2.0              2.0
148           6.2          3.4           5.0          2.3              2.0
149           5.9          3.0           5.0          1.8              2.0

     Species_0  Species_1  Species_2
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
..         …          …          …
145        0.0        0.0        1.0
146        0.0        0.0        1.0
147        0.0        0.0        1.0
148        0.0        0.0        1.0
149        0.0        0.0        1.0

[150 rows x 8 columns]
```

[ ]: