Data Preprocessing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format.

Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

Common steps involved in Data Preprocessing

1. Read the Dataset
2. Dealing with missing values
3. Dealing with categorical data
    - Ordinal feature mapping
    - Label Encoding
    - One-Hot Encoding
4. Feature Scaling
5. Splitting dataset into training and testing sets

Reading Dataset

pandas is a software library written for the Python programming language for easy and high performancedata manipulation and analysis

You can read data from a CSV file using the read_csv function. By default, it assumes that the fields are comma-separated.

```
import pandas as pd

#loading csv file using pandas
df = pd.read_csv('Weather.csv')

#Preview DataFrames with head()
df.head()
```

head.xcf

Dealing with missing values

For various reasons, many real world datasets contain missing values, often encoded as blanks, NaNs or other placeholders. But scikit-learn assume that all values in an array are numerical, and that all have and hold meaning. There is various ways to handle missing values of categorical ways.

1. Ignore observations of missing values if we are dealing with large data sets and less number of records has missing values
2. Ignore variable, if it is not significant
3. Replace with mean, meaian, mode(for neumeric data)
4. Replace by most frequent value(for categorical data)
5. Missing values can be treated as a separate category by itself(for categorical data)
6. Develop model to predict missing values

we need to check wether dataset contains missing values or not

```
#to check any columns contains missing vaues
df.isnull().any()
#to check count of missing vaues column wise
df.isnull().sum()
#to check count of missing vaues row wise
df.isnull().sum(axis = 1)
```

Deleting rows from dataframe

To delete rows and columns from DataFrames, Pandas uses the drop() function.

The drop() function in Pandas be used to delete rows from a DataFrame, with the axis set to 0.

```
"""droping all rows with missing cells via
using pandas.DataFrame.dropna()"""
#df.dropna()

#or some rows

X = df.iloc[:,:-1]
y = df.iloc[:,-1]
# Delete all rows with name 13
#X = X.drop(13, axis=0)
# Delete the rows with labels 0,1,4
X = X.drop([0,1,4], axis=0)
```

Deleting columns from dataframe

The drop() function in Pandas be used to delete columns from a DataFrame, with the axis set to 1.

```
# Delete column with name "Windy"
X = X.drop("Windy", axis=1)
# Delete multiple columns by providing list
#X = X.drop(["Windy","Humidity"], axis=1)
```

Imputation of missing values

The ploblem with the above two strategies is losing data which may be valuable (even though incomplete). A better strategy is to impute the missing values, i.e., to infer them from the known part of the data

Imputation of Neumeric Variables- Mean, Median and Mode

Computing the overall mean, median or mode is a very basic imputation method and it is very fast. Scikit Learn preprocessing contains a class called **Imputer** which will help us take care of the missing data.

```
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values="NaN", strategy="mean")
neumeric_values = X.iloc[:,1:3]
X.iloc[:,1:3] = imputer.fit_transform(neumeric_values)
```

Imputation of Categorical Variables

we can handle in two ways

1. Replace by most frequent value(Mode)
2. Missing values can be treated as a separate category by itself

```
#getting mode
md = X['Outlook'].mode()
#filling missing values with mode
X['Outlook'] = X['Outlook'].fillna(md[0])

# or

"""Missing values can be treated as a separate
category by itself here it is Unknown"""
X['Outlook'] = X['Outlook'].fillna('Unknown')
```

Dealing with categorical data

As we know scikit-learn models accepts only numerical data. If any categorical variables present in dataset we need to convert tem into neumerics

Even among categorical data, we may want to distinguish further between nominal and ordinal which can be sorted or ordered features. So, T-shirt size can be an ordinal feature, because we can define an order XL > L > M.

Ordinal feature mapping

we should convert the categorical string values into integers. However, since there is no convenient function that can automatically derive the correct order of the labels of our size feature, we have to define the mapping manually

```
color = ['green', 'red','blue']
size = ['M','L','XL']
price = [10.1, 13.5,15.3]
classLabel =['c1','c2','c1']
a = list(zip(color,size,price,classLabel))
df1 = pd.DataFrame(data = a,
columns = ["color","size","price","classLabel"])

size_mapping = {'M':1,'L':2,'XL':3}
df1['size']=df1['size'].map(size_mapping)
df1
size_mapping.items()
```

Label Encoding(Nominal feature encoding)

Label Encoding is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1.

Because scikit-learn's estimators treat class labels without any order, we can use LabelEncoder class to encode the string labels into integers.

```
#will not accept nan's
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
X.iloc[:, 0] = labelencoder_X.fit_transform(X.iloc[:, 0])
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
```

One Hot Encoding(Nominal feature encoding)

LabelEncoder tranforms categorical feature to one new feature of integers (0 to n_categories - 1) in our example Outlook values (sunny, rainy, Overcast) into (0,1,2) here 2(Overcast) is not greater than 0(sunny) Such integer representation can, however, not be used directly with all scikit-learn estimators, as these expect continuous input, and would interpret the categories as being ordered, which is often not desired

Another possibility to convert categorical features to features that can be used with scikit-learn estimators is to use a one-of-K, also known as one-hot or dummy encoding. This type of encoding can be obtained with the OneHotEncoder, which transforms each categorical feature with n_categories possible values into n_categories.

```
from sklearn.preprocessing import OneHotEncoder
onehot = OneHotEncoder(categorical_features = [0])
#make sure that X shoud have only neumeric values
X = onehot.fit_transform(X).toarray()
```

Feature scaling

Feature scaling is a method used to standardize the range of independent variables, it is also known as data normalization

Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

Another reason why feature scaling is applied is that gradient descent converges much faster with feature scaling than without it.

Feature scaling methods are

1. Standardization
2. Rescaling (min-max normalization)
3. Mean normalization
4. Scaling to unit length

Standardization

Standardization is widely used for normalization in many machine learning algorithms (e.g., support vector machines, logistic regression, and artificial neural networks). Feature standardization makes the values of each feature in the data have zero-mean (when subtracting the mean in the numerator) and unit-variance

standars scaler formula

min-max normalization

min_max scaler formula

Mean normalization

mean_norm formula

Unit Vector normalization

unit vector formula

We can use sklearn StandardScaler for normalize features

```
from sklearn.preprocessing import StandardScaler
sctandardscaler = StandardScaler()
X = sctandardscaler.fit_transform(X)
```

Feature Scaling Methods

Feature scaling is an essential step in data preprocessing for machine learning algorithms. Below are the most common methods:

1. Standardization (Z-score Normalization)

Standardization makes the feature values have zero mean and unit variance.

Formula: $X' = \frac{X-\mu}{\sigma}$

Where:

- μ = Mean of the feature
- σ = Standard deviation of the feature

Example:

Given the data: [10, 20, 30, 40, 50]

Mean (μ) = 30, Standard Deviation (σ) = 15.81

Original Value (X)	Standardized Value (X')
10	-1.26
20	-0.63
30	0.00
40	0.63
50	1.26

2. Min-Max Normalization

Scales the values between a given range (usually 0 to 1).

Formula: $X' = \frac{X-X_{\min}}{X_{\max}-X_{\min}}$

Example:

Given the data: [10, 20, 30, 40, 50]

Min ($X_{\min}$) = 10, Max ($X_{\max}$) = 50

Original Value (X)	Min-Max Scaled (X')
10	0.00
20	0.25
30	0.50
40	0.75
50	1.00

3. Mean Normalization

Scales the values between -1 and 1 using the mean and range of the data.

Formula: $X' = \frac{X-\mu}{X_{\max}-X_{\min}}$

Example:

Given the data: [10, 20, 30, 40, 50]

Mean (μ) = 30, Min = 10, Max = 50

Original Value (X)	Mean Normalized (X')
10	-0.67
20	-0.33
30	0.00
40	0.33
50	0.67

4. Scaling to Unit Length (Unit Vector Normalization)

Scales the values so that the vector has a norm (length) of 1.

Formula: $X' = \frac{X}{||X||}$

Using L2 norm: $||X||_2 = \sqrt{\sum X^2}$

Example:

Given the data: [3, 4]

Norm ($||X||_2$) = $\sqrt{3^2 + 4^2} = 5$

Original Value (X)	Unit Vector (X')
3	0.6
4	0.8

Implementation using Scikit-learn

Standardization:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Min-Max Normalization:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

Mean Normalization:

```
X_scaled = (X - X.mean(axis=0)) / (X.max(axis=0) - X.min(axis=0))
```

Unit Vector Normalization:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
X_normalized = normalizer.fit_transform(X)
```

These methods help improve the performance of machine learning models by ensuring consistent feature scaling.