Kmit

SONET

Data Science
(Level-I)

Data Preprocessing

Kmit SONET

Steps for Machine Learning

Machine Learning

Look at the big picture.

Get the data.

Discover and visualize the data to gain insights.

Prepare the data for Machine Learning algorithms. — Numpy, pandas, SkLearn

Select a model and train it.
Train the model

Fine-tune your model.

Present your solution.

Launch, monitor, and maintain your system

Introduction

Why preprocessing ?

Real world data are generally

Incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data

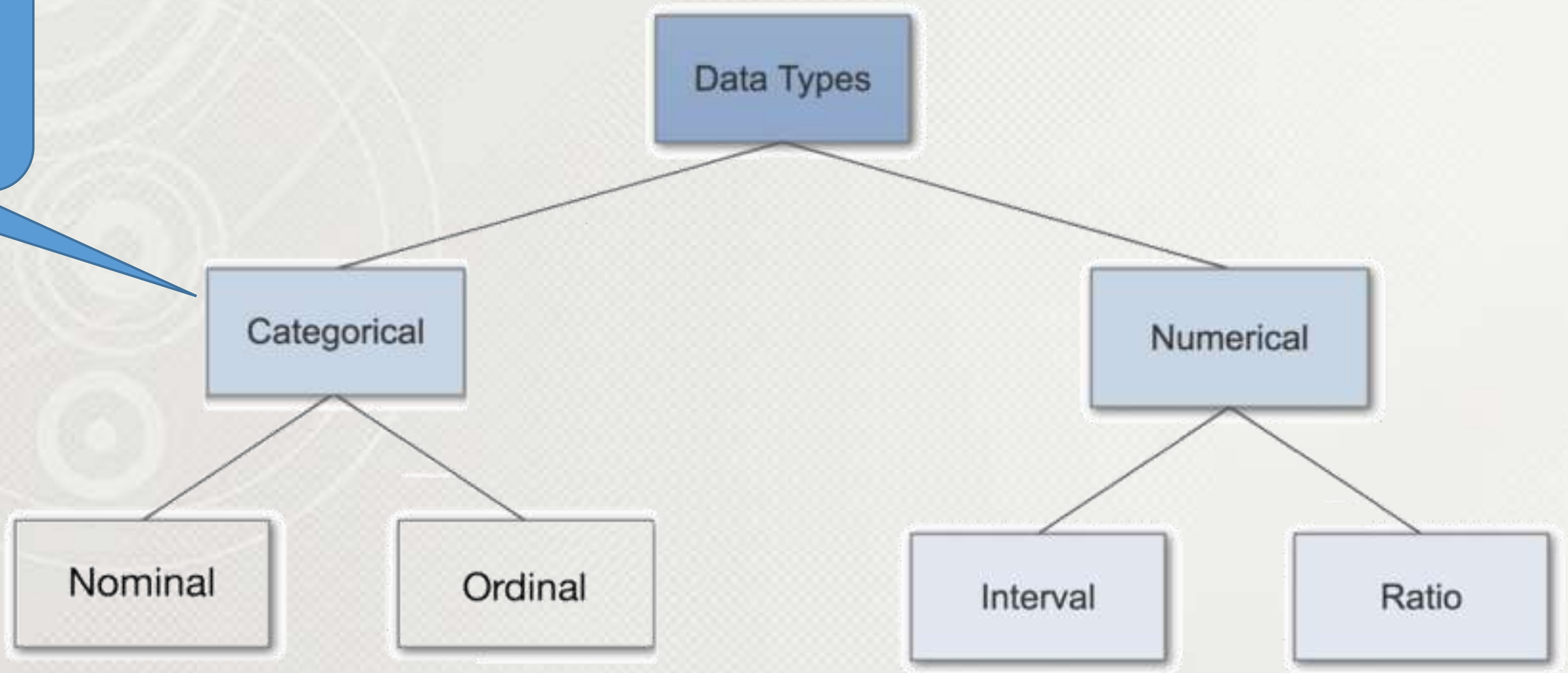
Noisy: containing errors or outliers

Inconsistent: containing discrepancies in codes or names Tasks in data preprocessing

Introduction

- Data pre-processing is an important step of solving every machine learning problem.
- Most of the datasets used with Machine Learning problems need to be processed / cleaned / transformed so that a Machine Learning algorithm can be trained on it.
- Most commonly used preprocessing techniques are very few like - missing value imputation, encoding categorical variables, scaling, etc.

Types of Data



Types of Data

Categorical data:

- It represents characteristics.
- Therefore it can represent things like a person's gender, language etc.

1. **Nominal Data**

Nominal values represent discrete units and are used to label variables, that have no quantitative value. Just think of them as „labels".

➢ Note that nominal data that has no order.

EX: what's your favourite movie

- Spider man
- Ant man
- Iron man

Types of Data

2.Ordinal Data

Ordinal values represent discrete and ordered units. It is therefore nearly the same as nominal data, “except that it’s ordering matters”.

Ex:

What’s your rating for Avengers Infinity War?

- *
- **
- ***
- ****
- *****

Types of Data

2. Numerical Data

2.1 Interval Data

Interval values represent ordered units that have the same difference.

Ex: Temparature?

- -10
- -5
- 0
- 5
- 10
- 15

2.2 Ratio Data

Ratio values are ordered units with intermediate values. Ratio values are the same as interval values, with the difference that they do have an absolute zero.

Ex: height, weight…

Handling Numeric data

- Too many nulls - When most (over 60% to 70%) of the values in a column are null, it's better to drop the column.
- Same values/skew - Sometimes, a majority of values in a column might be same values with very few different values. We need to check if the occurrence of such values is due to a skew in dataset or is it natural for that dataset. If it's skewed, dataset should be resampled (sub-sample or over-sample, as appropriate). If it's not a skew and the values occur naturally in that way, it's better to drop the column.

Handling Numeric data

- **Data types** - Check the datatypes of the columns, particularly date columns and type cast appropriately.
- **Missing value imputation** - Usually median is used with numeric columns and mode with non-numeric columns.
- **When column doesn’t have missing values** - It’s possible that a column doesn’t have any null values in the train dataset, but it’s very possible that it might have null values in test dataset. Hence, it’s important to review the columns/data and perform missing value imputation of all columns that can possibly have missing values, even if the train dataset doesn’t have any missing values.

Handling Numerical data

1. Load DataSet

```
In [29]: import pandas as pd
         data=pd.read_csv("C:\\Users\\Venkatesh\\Desktop\\datasets\\fishing.csv")
         print(data.head())
```

```
   site  totabund   density  meandepth  year    sweptarea
0   1.0        76  0.002070        804  1978          NaN
1   2.0       161  0.003520        808  2001  45741.25391
2   3.0        39  0.000981        809  2001  39775.00000
3   NaN       410  0.008039        848  1979  51000.00000
4   5.0       177  0.005933        853  2002  29831.25195
```

Handling Numerical data

2. Check is there any null values

```
In [30]: data.isnull().sum()

Out[30]: site          10
         totabund       0
         density        9
         meandepth      0
         year           0
         sweptarea      1
         dtype: int64
```

3. fillna() method –filling null values with ‘1’

```
In [31]: df=data.fillna(1)
         print(df.head())
```

	site	totabund	density	meandepth	year	sweptarea
0	1.0	76	0.002070	804	1978	1.00000
1	2.0	161	0.003520	808	2001	45741.25391
2	3.0	39	0.000981	809	2001	39775.00000
3	1.0	410	0.008039	848	1979	51000.00000
4	5.0	177	0.005933	853	2002	29831.25195

Handling Numerical data

In [32]:
```
df=data.fillna(data.mean())
print(df.head())
```

	site	totabund	density	meandepth	year	sweptarea
0	1.000000	76	0.002070	804	1978	64992.287498
1	2.000000	161	0.003520	808	2001	45741.253910
2	3.000000	39	0.000981	809	2001	39775.000000
3	76.029197	410	0.008039	848	1979	51000.000000
4	5.000000	177	0.005933	853	2002	29831.251950

Filling missing data with mean

In [33]:
```
df=data.fillna(data.mode())
print(df.head())
```

	site	totabund	density	meandepth	year	sweptarea
0	1.0	76	0.002070	804	1978	59662.50391
1	2.0	161	0.003520	808	2001	45741.25391
2	3.0	39	0.000981	809	2001	39775.00000
3	5.0	410	0.008039	848	1979	51000.00000
4	5.0	177	0.005933	853	2002	29831.25195

Filling missing data with mode

Handling Numerical data

Dropping all null values

```
In [10]: data.dropna(inplace=True)
         data.isnull().sum()
         #print(data.head())

Out[10]: site         0
         totabund     0
         density      0
         meandepth    0
         year         0
         sweptarea    0
         dtype: int64
```

Outliers

An *outlier* is an observation that lies an abnormal distance from other values in a random sample from a population.

To remove outliers we will use

- np.log() Natural logarithm, element-wise.
- np.sqrt() square root
- np.cbrt() cube root

Handling Outliers

Check the difference between minimum value and maximum value

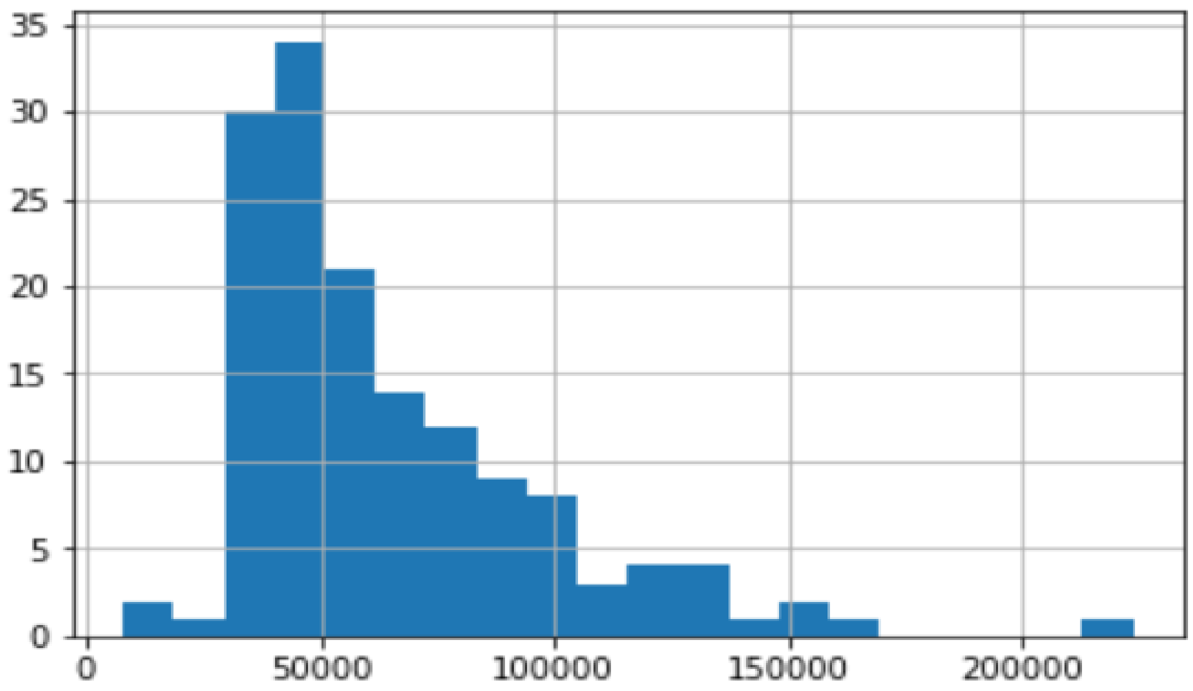
```
In [17]: df['sweptarea'].max()        # max = 223440.0
         df['sweptarea'].min()        # min = 7970.0
         df['sweptarea'].mean()       # mean = 64956.030466666685

Out[17]: 223440.0
```

Handling Outliers

Histogram for original ‘sweptarea’ data

```
In [15]: df['sweptarea'].hist(bins=20)
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1b14cbb52e8>
```

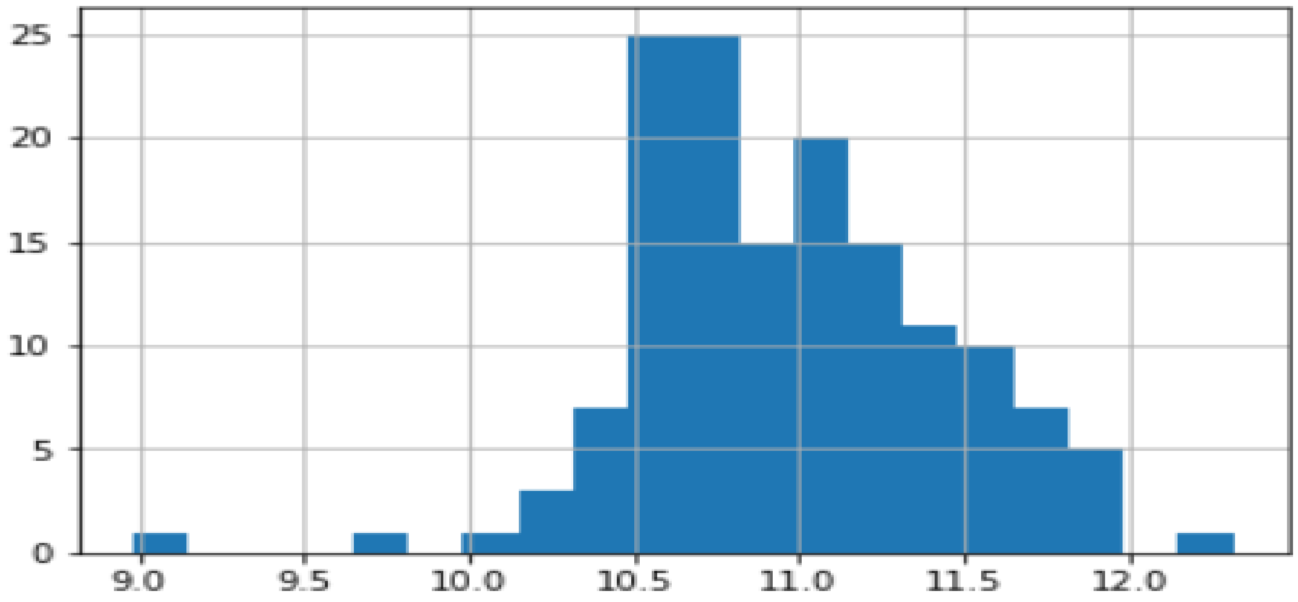


Handling Outliers

Handling outliers with numpy.log():

This mathematical function helps user to calculate **Natural logarithm of x** where x belongs to all the input elements.

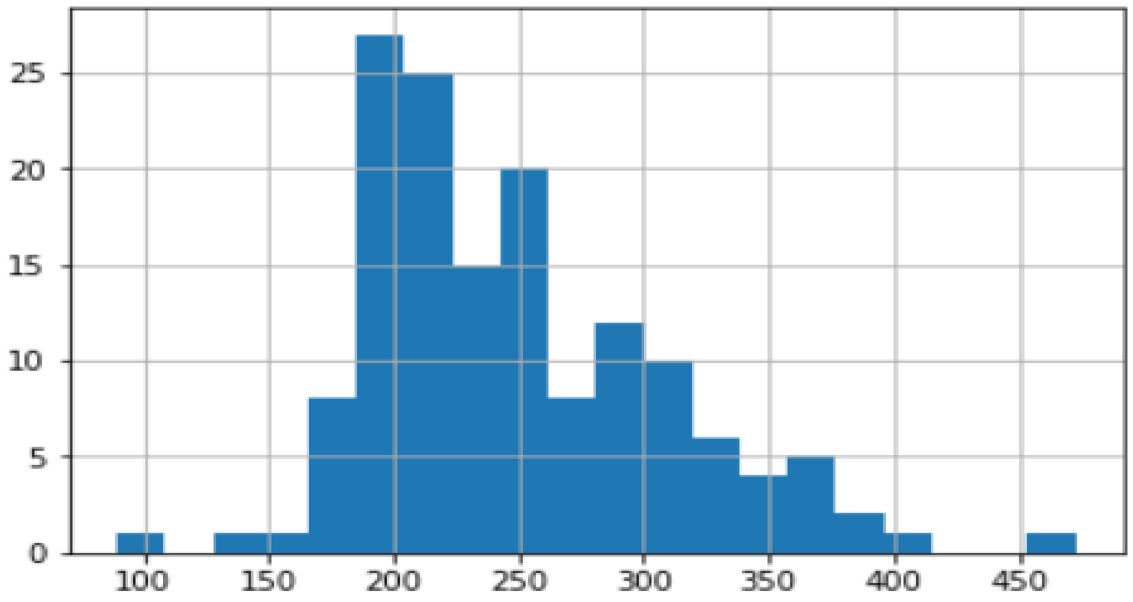
```
In [12]:  import numpy as np
          df['sweptarea_log']=np.log(df['sweptarea'])
          df['sweptarea_log'].hist(bins=20)

Out[12]:  <matplotlib.axes._subplots.AxesSubplot at 0x1b14ca61080>
```



Handling Outliers

numpy.sqrt():
Return the positive square-root of an array(data), element-wise.

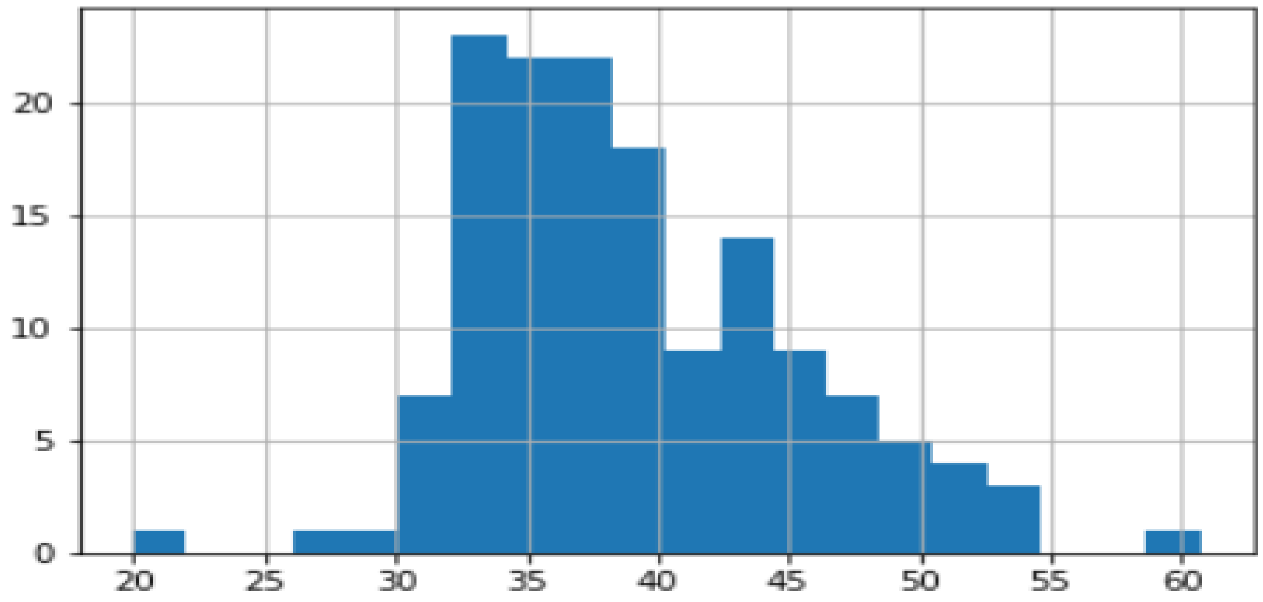
```
In [13]: df['sweptarea_sqrt']=np.sqrt(df['sweptarea'])
         df['sweptarea_sqrt'].hist(bins=20)

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1b14cacce10>
```



Handling Outliers

numpy.cbrt():
This mathematical function helps user to calculate cube root of x for all x being the array elements.

```
In [14]: df['sweptarea_cbrt']=np.cbrt(df['sweptarea'])
         df['sweptarea_cbrt'].hist(bins=20)

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1b14c9eeba8>
```



Observe the data after applying different functions

```
In [18]: print(df.head(5))

   site  totabund   density  meandepth  year   sweptarea  sweptarea_log  \
0   1.0        76  0.002070        804  1978  59662.50391      10.996459
1   2.0       161  0.003520        808  2001  45741.25391      10.730756
2   3.0        39  0.000981        809  2001  39775.00000      10.590994
3   5.0       410  0.008039        848  1979  51000.00000      10.839581
4   5.0       177  0.005933        853  2002  29831.25195      10.303312

   sweptarea_sqrt  sweptarea_cbrt
0      244.259092       39.075135
1      213.872050       35.763171
2      199.436707       34.135274
3      225.831796       37.084298
4      172.717260       31.013956
```

Normalization

It is the process of reorganizing data in a dataset so that it meets two basic requirements:
(1)There is no redundancy of data (all data is stored in only one place), and
(2)data dependencies are logical (all related data items are stored together)

Normalizing in scikit-learn refers to rescaling each observation (row) to have a length of 1 (called a unit norm in linear algebra).
This preprocessing can be useful for sparse datasets (lots of zeros) with attributes of varying scales when using algorithms that weight input values such as neural networks and algorithms that use distance measures such as K-Nearest Neighbors.

Normalization

```
In [33]: from sklearn import preprocessing
         dfnorm=preprocessing.normalize(df)
         print(dfnorm)

[[1.53780127e-05 1.16872897e-03 3.18368076e-08 ... 1.70419504e-04
  3.92040674e-03 6.18283263e-04]
 [4.36748663e-05 3.51582674e-03 7.68633754e-08 ... 2.34332164e-04
  4.67041661e-03 7.80975861e-04]
 [7.53124490e-05 9.79061837e-04 2.46149953e-08 ... 2.65877895e-04
  5.00668893e-03 8.56937033e-04]
 ...
 [1.60348919e-03 2.63587264e-04 2.89974545e-09 ... 1.25396317e-04
  3.31127402e-03 4.93819014e-04]
 [1.05908181e-03 2.73776251e-04 1.97387633e-09 ... 8.53034103e-05
  2.68318369e-03 3.72938880e-04]
 [1.10444705e-03 2.16411922e-04 1.61622393e-09 ... 8.80934469e-05
  2.73069675e-03 3.81777248e-04]]
```

Categorical data

Categorical Attributes –

- When the number unique values in a categorical column are too high, check the value counts of each of those values. Replace rarely occurring values together into a single value like ‘Other’ before encoding.
- When number of unique values is huge and even the values are equally distributed, try to find some related values and see if the multiple categorical values can be clubbed into single (grouping), thereby reducing the count of categorical values.

Related Attributes –

- If there multiple attributes with same information with different granularity, like city and state, it’s better to keep columns like state and delete city column. Additionally, keeping both columns and assessing feature importance might help in eliminating one column.

Handling Categorical data

1.Label encoding

2.Range encoding

3.one-hot encoding

Handling Categorical data

1.Label encoding

Sex	New_sex
Male	1
Female	0
Female	0
Male	1
Male	1

2.Range encoding

Height	Avg_Height	Low_Height	High_Height
100-110	105	100	100
110-120	115	110	110
120-130	125	120	120
130-140	135	130	130
140-150	145	140	150

Handling Categorical data

2.one-hot encoding

Hyderabad
Guntur
Hyderabad
Vizag
Vizag

city	New_city_hyd	New_city_gnt	New_city_viz
Hyderabad	1	0	0
Guntur	0	1	0
Hyderabad	1	0	0
Vizag	0	0	1
Vizag	0	0	1

Categorical data

Label Encoder	One Hot Encoder
Numeric representation, ordinals	Binary representation
Loses uniqueness of values, single dimension in vector space	Individual values expressed as a different dimension in orthogonal vector space
Suitable with categorical values that are ordinal in nature, like – fog_level (low, medium, high)	Suitable with non-ordinal types of categorical attributes, like – car_type (hatchback, sedan, SUV, etc.)
Label encoded categorical attributes don't pose any further challenges	One hot encoded categorical attributes might dramatically increase the feature space (curse of dimensionality). When One hot encoding is used, it's often followed by PCA to tackle high-dimensionality


THANK
YOU