

**BOWEN UNIVERSITY, IWO**

**COLLEGE OF COMPUTING AND COMMUNICATION STUDIES**

**DEPARTMENT OF COMPUTER SCIENCE**

**CSC 404- MACHINE LEARNING WITH PYTHON**

**LECTURE NOTE 6**

Understanding Data with Statistics

**Introduction**

While working with machine learning projects, usually we ignore two most important parts called ***mathematics*** and ***data***. It is because, we know that ML is a data driven approach and our ML model will produce only as good or as bad results as the data we provided to it.

In the previous chapter, we discussed how we can upload CSV data into our ML project, but it would be good to understand the data before uploading it. We can understand the data by two ways, with statistics and with visualization.

In this chapter, with the help of following Python recipes, we are going to understand ML data with statistics.

**Looking at Raw Data**

The very first recipe is for looking at your raw data. It is important to look at raw data because the insight we will get after looking at raw data will boost our chances to better pre-processing as well as handling of data for ML projects.

Following is a Python script implemented by using head() function of Pandas DataFrame on Pima Indians diabetes dataset to look at the first 50 rows to get better understanding of it:

**Example**

from pandas import read\_csv

path = r"C:\pima-indians-diabetes.csv"

headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data = read\_csv(path, names=headernames)

print(data.head(50))

**Output**

preg plas pres skin test mass pedi age class

0 6 148 72 35 0 33.6 0.627 50 1

1 1 85 66 29 0 26.6 0.351 31 0

2 8 183 64 0 0 23.3 0.672 32 1

3 1 89 66 23 94 28.1 0.167 21 0

4 0 137 40 35 168 43.1 2.288 33 1

5 5 116 74 0 0 25.6 0.201 30 0

6 3 78 50 32 88 31.0 0.248 26 1

7 10 115 0 0 0 35.3 0.134 29 0

8 2 197 70 45 543 30.5 0.158 53 1

9 8 125 96 0 0 0.0 0.232 54 1

10 4 110 92 0 0 37.6 0.191 30 0

11 10 168 74 0 0 38.0 0.537 34 1

12 10 139 80 0 0 27.1 1.441 57 0

13 1 189 60 23 846 30.1 0.398 59 1

14 5 166 72 19 175 25.8 0.587 51 1

15 7 100 0 0 0 30.0 0.484 32 1

16 0 118 84 47 230 45.8 0.551 31 1

17 7 107 74 0 0 29.6 0.254 31 1

18 1 103 30 38 83 43.3 0.183 33 0

19 1 115 70 30 96 34.6 0.529 32 1

20 3 126 88 41 235 39.3 0.704 27 0

21 8 99 84 0 0 35.4 0.388 50 0

22 7 196 90 0 0 39.8 0.451 41 1

23 9 119 80 35 0 29.0 0.263 29 1

24 11 143 94 33 146 36.6 0.254 51 1

25 10 125 70 26 115 31.1 0.205 41 1

26 7 147 76 0 0 39.4 0.257 43 1

27 1 97 66 15 140 23.2 0.487 22 0

28 13 145 82 19 110 22.2 0.245 57 0

29 5 117 92 0 0 34.1 0.337 38 0

30 5 109 75 26 0 36.0 0.546 60 0

31 3 158 76 36 245 31.6 0.851 28 1

32 3 88 58 11 54 24.8 0.267 22 0

33 6 92 92 0 0 19.9 0.188 28 0

34 10 122 78 31 0 27.6 0.512 45 0

35 4 103 60 33 192 24.0 0.966 33 0

36 11 138 76 0 0 33.2 0.420 35 0

37 9 102 76 37 0 32.9 0.665 46 1

38 2 90 68 42 0 38.2 0.503 27 1

39 4 111 72 47 207 37.1 1.390 56 1

40 3 180 64 25 70 34.0 0.271 26 0

41 7 133 84 0 0 40.2 0.696 37 0

42 7 106 92 18 0 22.7 0.235 48 0

43 9 171 110 24 240 45.4 0.721 54 1

44 7 159 64 0 0 27.4 0.294 40 0

45 0 180 66 39 0 42.0 1.893 25 1

46 1 146 56 0 0 29.7 0.564 29 0

47 2 71 70 27 0 28.0 0.586 22 0

48 7 103 66 32 0 39.1 0.344 31 1

49 7 105 0 0 0 0.0 0.305 24 0

We can observe from the above output that first column gives the row number which can be very useful for referencing a specific observation.

**Checking Dimensions of Data**

It is always a good practice to know how much data, in terms of rows and columns, we are having for our ML project. The reasons behind are:

* Suppose if we have too many rows and columns then it would take long time to run the algorithm and train the model.
* Suppose if we have too less rows and columns then it we would not have enough data to well train the model.

Following is a Python script implemented by printing the **shape** property on Pandas Data Frame. We are going to implement it on iris data set for getting the total number of rows and columns in it.

**Example**

from pandas import read\_csv

path = r"C:\iris.csv"

data = read\_csv(path)

print(data.shape)

**Output**

(150, 4)

We can easily observe from the output that iris data set, we are going to use, is having 150 rows and 4 columns.

**Getting Each Attribute’s Data Type**

It is another good practice to know data type of each attribute. The reason behind is that, as per to the requirement, sometimes we may need to convert one data type to another. For example, we may need to convert string into floating point or int for representing categorial or ordinal values. We can have an idea about the attribute’s data type by looking at the raw data, but another way is to use **dtypes** property of Pandas DataFrame. With the help of **dtypes** property we can categorize each attributes data type. It can be understood with the help of following Python script:

**Example**

from pandas import read\_csv

path = r"C:\iris.csv"

data = read\_csv(path)

print(data.dtypes)

**Output**

sepal\_length float64

sepal\_width float64

petal\_length float64

petal\_width float64

dtype: object

From the above output, we can easily get the datatypes of each attribute.

**Statistical Summary of Data**

We have discussed Python recipe to get the shape i.e. number of rows and columns, of data but many times we need to review the summaries out of that shape of data. It can be done with the help of **describe()** function of Pandas DataFrame that further provide the following 8 statistical properties of each & every data attribute:

Count

Mean

Standard Deviation

Minimum Value

Maximum value

25%

Median i.e. 50%

75%

**Example**

from pandas import read\_csv

from pandas import set\_option

path = r"C:\pima-indians-diabetes.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data = read\_csv(path, names=names)

set\_option('display.width', 100)

set\_option('precision', 2)

print(data.shape)

print(data.describe())

**Output**

(768, 9)

preg plas pres skin test mass pedi age class

count 768.00 768.00 768.00 768.00 768.00 768.00 768.00 768.00 768.00

mean 3.85 120.89 69.11 20.54 79.80 31.99 0.47 33.24 0.35

std 3.37 31.97 19.36 15.95 115.24 7.88 0.33 11.76 0.48

min 0.00 0.00 0.00 0.00 0.00 0.00 0.08 21.00 0.00

25% 1.00 99.00 62.00 0.00 0.00 27.30 0.24 24.00 0.00

50% 3.00 117.00 72.00 23.00 30.50 32.00 0.37 29.00 0.00

75% 6.00 140.25 80.00 32.00 127.25 36.60 0.63 41.00 1.00

max 17.00 199.00 122.00 99.00 846.00 67.10 2.42 81.00 1.00

From the above output, we can observe the statistical summary of the data of Pima Indian Diabetes dataset along with shape of data.

**Reviewing Class Distribution**

Class distribution statistics is useful in classification problems where we need to know the balance of class values. It is important to know class value distribution because if we have highly imbalanced class distribution i.e. one class is having lots more observations than other class, then it may need special handling at data preparation stage of our ML project. We can easily get class distribution in Python with the help of Pandas DataFrame.

**Example**

from pandas import read\_csv

path = r"C:\pima-indians-diabetes.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data = read\_csv(path, names=names)

count\_class = data.groupby('class').size()

print(count\_class)

**Output:**

Class

0 500

1 268

dtype: int64

From the above output, it can be clearly seen that the number of observations with class 0 are almost double than number of observations with class 1.

**Reviewing Correlation between Attributes**

The relationship between two variables is called correlation. In statistics, the most common method for calculating correlation is Pearson’s Correlation Coefficient. It can have three values as follows:

* **Coefficient value = 1:** It represents full **positive** correlation between variables.
* **Coefficient value = -1:** It represents full **negative** correlation between variables.
* **Coefficient value = 0:** It represents **no** correlation at all between variables.

It is always good for us to review the pairwise correlations of the attributes in our dataset before using it into ML project because some machine learning algorithms such as linear regression and logistic regression will perform poorly if we have highly correlated attributes. In Python, we can easily calculate a correlation matrix of dataset attributes with the help of **corr()** function on Pandas DataFrame.

**Example**

from pandas import read\_csv

from pandas import set\_option

path = r"C:\pima-indians-diabetes.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data = read\_csv(path, names=names)

set\_option('display.width', 100)

set\_option('precision', 2)

correlations = data.corr(method='pearson')

print(correlations)

**Output**

preg plas pres skin test mass pedi age class

preg 1.00 0.13 0.14 -0.08 -0.07 0.02 -0.03 0.54 0.22

plas 0.13 1.00 0.15 0.06 0.33 0.22 0.14 0.26 0.47

pres 0.14 0.15 1.00 0.21 0.09 0.28 0.04 0.24 0.07

skin -0.08 0.06 0.21 1.00 0.44 0.39 0.18 -0.11 0.07

test -0.07 0.33 0.09 0.44 1.00 0.20 0.19 -0.04 0.13

mass 0.02 0.22 0.28 0.39 0.20 1.00 0.14 0.04 0.29

pedi -0.03 0.14 0.04 0.18 0.19 0.14 1.00 0.03 0.17

age 0.54 0.26 0.24 -0.11 -0.04 0.04 0.03 1.00 0.24

class 0.22 0.47 0.07 0.07 0.13 0.29 0.17 0.24 1.00

The matrix in above output gives the correlation between all the pairs of the attribute in dataset.

**Reviewing Skew of Attribute Distribution**

Skewness may be defined as the distribution that is assumed to be Gaussian but appears distorted or shifted in one direction or another, or either to the left or right. Reviewing the skewness of attributes is one of the important tasks due to following reasons:

* Presence of skewness in data requires the correction at data preparation stage so that we can get more accuracy from our model.
* Most of the ML algorithms assumes that data has a Gaussian distribution i.e. either normal of bell curved data.

In Python, we can easily calculate the skew of each attribute by using **skew()** function on Pandas DataFrame.

**Example**

from pandas import read\_csv

path = r"C:\pima-indians-diabetes.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data = read\_csv(path, names=names)

print(data.skew())

**Output**

preg 0.90

plas 0.17

pres -1.84

skin 0.11

test 2.27

mass -0.43

pedi 1.92

age 1.13

class 0.64

dtype: float64

From the above output, positive or negative skew can be observed. If the value is closer to zero, then it shows less skew.