Bentley University MA346 in Python

Content extracted from the How to Data Website

This PDF generated on 18 January 2024

Contents

MA346 is an undergraduate data science course at Bentley University. The description from the course catalog can be found here. Topics included in the course are listed as tasks (on website) below.

Mathematical topics include functions and relations, a review of basic statistics, and (time permitting) networks, matrices, and an introduction to supervised learning.

Computing topics include Jupyter notebooks (locally and in the cloud), Python and pandas, abstraction, concatenation and merging, map-reduce, split-apply-combine, data munging, version control, and dashboards.

Communication topics include best practices for writing reports, documenting code and computational notebooks, and data visualization.

This page is not yet complete. More content will be added here over time.

Content last modified on 03 August 2023.

How to do basic mathematical computations

Description

How do we write the most common mathematical operations in a given piece of software? For example, how do we write multiplication, or exponentiation, or logarithms, in Python vs. R vs. Excel, and so on?

Solution in Python using NumPy

This answer assumes you have imported NumPy as follows.

```
import numpy as np
```

Mathematical notation	Python code	Requires NumPy?
$x+y$	`x+y`	no
$x-y$	`x-y`	no
xy	`x*y`	no
$\frac{x}{y}$	`x/y`	no
$\left\lfloor\frac{x}{y}\right\rfloor$	`x//y`	no
$\left\lfloor\frac{x}{y}\right\rfloor$	`np.floor_divide(x,y)`	yes
remainder of $x \div y$	`x%y`	no
remainder of $x \div y$	`np.remainder(x,y)`	yes
x^y	`x**y`	no
$\|x\|$	`abs(x)`	no
$\|x\|$	`np.abs(x)`	yes
$\ln x$	`np.log(x)`	yes
$\log_a b$	`np.log(b)/np.log(a)`	yes
e^x	`np.exp(x)`	yes
π	`np.pi`	yes
$\sin x$	`np.sin(x)`	yes
$\sin^{-1} x$	`np.asin(x)`	yes
$\sqrt{x}$	`x**0.5`	no
$\sqrt{x}$	`np.sqrt(x)`	yes

Other trigonometric functions are also available besides just `np.sin`, including `np.cos`, `np.tan`, etc.

NumPy automatically applies any of these functions to all entries of a NumPy array or pandas Series, but the built-in Python functions do not have this feature. For example, to square all numbers in an array, see below.

```
import numpy as np
example_array = np.array( [ -3, 2, 0.5, -1, 10, 9.2, -3.3 ] )
example_array ** 2
```

```
array([  9.  ,   4.  ,   0.25,   1.  , 100.  ,  84.64,  10.89])
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to quickly load some sample data

Description

Sometimes you just need to try out a new piece of code, whether it be data manipulation, statistical computation, plotting, or whatever. And it's handy to be able to quickly load some example data to work with. There is a lot of freely available sample data out there. What's the easiest way to load it?

Solution in pure Python

The R programming language comes with many free datasets built in. To make these same datasets available to Python programmers as well, you can install and import the `rdatasets` package.

First, ensure that you have it installed, by running `pip install rdatasets` or `conda install rdatasets` from your command line. Then you can get access to many datasets as follows:

```
from rdatasets import data
df = data( 'iris' )  # Load the famous Fisher's irises dataset
df.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

But what datasets are available? There are many! You can find a full list in the package itself.

```
from rdatasets import summary
summary()
```

	Package	Item	Title	Rows	Cols	n_binary	n_chara
0	boot	acme	Monthly Excess Returns	60	3	0	1
1	boot	aids	Delay in AIDS Reporting in England and Wales	570	6	1	0
2	boot	aircondit	Failures of Air-conditioning Equipment	12	1	0	0
3	boot	aircondit7	Failures of Air-conditioning Equipment	24	1	0	0
4	boot	amis	Car Speeding and Warning Signs	8437	4	1	0
...	...	...	...	...	...	...	...
1340	Zelig	tobin	Tobin's Tobit Data	20	3	0	0
1341	Zelig	turnout	Turnout Data Set from the National Election Su...	2000	5	2	0
1342	Zelig	voteincome	Sample Turnout and Demographic Data from the 2...	1500	7	3	0
1343	Zelig	Weimar	1932 Weimar election data	10	11	0	0
1344	Zelig	Zelig.url	Table of links for Zelig	49	2	0	0

1345 rows × 12 columns

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to compute summary statistics

Description

The phrase "summary statistics" usually refers to a common set of simple computations that can be done about any dataset, including mean, median, variance, and some of the others shown below.

Related tasks:

- How to summarize a column (on website)
- How to summarize and compare data by groups (on website)

Solution in Python using pandas and NumPy

We first load a famous dataset, Fisher's irises, just to have some example data to use in the code that follows. (See how to quickly load some sample data.)

```
from rdatasets import data
df = data( 'iris' )
```

How big is the dataset? The output shows number of rows then number of columns.

```
df.shape
```

```
(150, 5)
```

What are the columns and their data types? Are any values missing?

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Sepal.Length  150 non-null    float64
 1   Sepal.Width   150 non-null    float64
 2   Petal.Length  150 non-null    float64
 3   Petal.Width   150 non-null    float64
 4   Species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

What do the first few rows look like?

```
df.head()  # Default is 5, but you can do df.head(20) or any number.
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

The easiest way to get summary statistics for a pandas DataFrame is with the `describe` function.

```
df.describe()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

The individual statistics are the row headings, and the numeric columns from the original dataset are listed across the top.

We can also compute these statistics (and others) one at a time for any given set of data points. Here, we let `xs` be one column from the above DataFrame, but you could use any NumPy array or pandas DataFrame instead.

```
xs = df['Sepal.Length']

import numpy as np

np.mean( xs )           # mean, or average, or center of mass
np.median( xs )         # 50th percentile
np.percentile( xs, 25 ) # compute any percentile, such as the 25th
np.var( xs )            # variance
np.std( xs )            # standard deviation, the square root of the variance
np.sort( xs )           # data in increasing order
np.sum( xs )            # sum, or total
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to convert a text column into dates

Description

When loading data, many software systems make intelligent guesses about the format and data type of each column, but sometimes that is not sufficient. If you have a column of text that should be interpreted as dates, how can we ask the software to convert it?

Solution in Python using pandas

Let's create a small example DataFrame to use here (using the method from how to create a data frame from scratch). Naturally, you would apply this solution to your own data instead.

```
import pandas as pd
df = pd.DataFrame( { 'Date' :  [ '5/7/19', '5/10/19',   '5/11/19' ],
                     'Event' : [   'Work',   'Party', 'More work' ] } )
df
```

	Date	Event
0	5/7/19	Work
1	5/10/19	Party
2	5/11/19	More work

If you've already got the data in a DataFrame column, and you wish to convert it to dates, use the `pd.to_datetime` function, which will do its best to read whatever format your dates are in:

```
df['Date'] = pd.to_datetime( df['Date'] )
df
```

```
/tmp/ipykernel_4832/1311655736.py:1: UserWarning: Could not infer format, so each element will be parsed individual
  df['Date'] = pd.to_datetime( df['Date'] )
```

	Date	Event
0	2019-05-07	Work
1	2019-05-10	Party
2	2019-05-11	More work

But if they aren't in a standard format, you can specify just about any format as in the following example. See the Python documentation for format details.

```
# If the dates had been, for example, 5-7-2019 10:15:00
df['Date'] = pd.to_datetime( df['Date'], format="%m-%d-%Y %H:%M:%S" )
```

It's often easier to handle date conversions while reading the data. You can tell pandas to read dates in most of the common date formats using any of the following methods.

```python
# Any columns that look like dates, treat as dates:
df = pd.read_csv( "example.csv", parse_dates=True )

# Convert the specific columns you name into dates:
df = pd.read_csv( "example.csv", parse_dates=['col1','col2'] )

# If the date is spread over multiple columns, do this:
# (Let's say the year, month, and day are in columns, 4, 5, and 6.)
df = pd.read_csv( "example.csv", parse_dates=[[4,5,6]] )
# Note the double brackets, and indices start counting at zero.
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to create a data frame from scratch

Description

Sometimes it is useful to create a small table of data directly in code, without first needing to store the data in a file and load it from there. This can be useful for creating small tables for testing purposes, or for creating small lookup tables that hold abbreviations, IDs, etc. What's the easiest way to build such a table?

Solution in pure Python

In pandas, the `pd.DataFrame` function can construct new DataFrames. Just provide it with a dictionary whose keys are the column headers and whose values are the column contents.

Here's an example:

```python
import pandas as pd
df = pd.DataFrame( {
    'Last name' :  [ 'Potter', 'Weasley', 'Granger', 'Malfoy' ],
    'First name' : [ 'Harry', 'Ron', 'Hermione', 'Draco' ],
    'House' :      [ 'Gryffindor', 'Gryffindor', 'Gryffindor', 'Slytherin' ]
} )
df
```

	Last name	First name	House
0	Potter	Harry	Gryffindor
1	Weasley	Ron	Gryffindor
2	Granger	Hermione	Gryffindor
3	Malfoy	Draco	Slytherin

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to compute covariance and correlation coefficients

Description

Covariance is a measure of how much two variables "change together." It is positive when the variables tend to increase or decrease together, and negative when they upward motion of one variable is correlated with downward motion of the other. Correlation normalizes covariance to the interval $[-1, 1]$.

Solution in Python using pandas and NumPy

We will construct some random data here, but when applying this, you would use your own data, of course.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(10,5))
df.columns = [ 'col1','col2','col3','col4','col5' ]
df.head()
```

	col1	col2	col3	col4	col5
0	0.488293	0.151749	0.485939	0.278562	0.998647
1	0.405459	0.766983	0.915349	0.099784	0.518523
2	0.312085	0.498104	0.526030	0.745883	0.292882
3	0.313217	0.826840	0.254793	0.942009	0.456271
4	0.657147	0.024847	0.769884	0.140779	0.427270

If you have two pandas Series, you can compute the covariance of just those two variables. Note that every column in a DataFrame is a pandas series.

```
np.cov( df['col1'], df['col2'] )
```

```
array([[ 0.04524431, -0.02545402],
       [-0.02545402,  0.12901528]])
```

You can also compare all of a DataFrame's columns among one another, each as a separate variable.

```
df.cov()
```

	col1	col2	col3	col4	col5
col1	0.045244	-0.025454	0.005095	-0.015552	-0.006827
col2	-0.025454	0.129015	0.009857	0.062661	-0.013753
col3	0.005095	0.009857	0.084701	-0.048114	0.014510
col4	-0.015552	0.062661	-0.048114	0.087198	-0.023934
col5	-0.006827	-0.013753	0.014510	-0.023934	0.057866

The Pearson correlation coefficient can be computed with `np.corrcoef` in place of `np.cov`.

```
np.corrcoef( df['col1'], df['col2'] )
```

```
array([[ 1.        , -0.33316075],
       [-0.33316075,  1.        ]])
```

And pandas DataFrames have a built in method to do this for all numeric columns.

```
df.corr()
```

	col1	col2	col3	col4	col5
col1	1.000000	-0.333161	0.082300	-0.247604	-0.133423
col2	-0.333161	1.000000	0.094296	0.590780	-0.159177
col3	0.082300	0.094296	1.000000	-0.559850	0.207259
col4	-0.247604	0.590780	-0.559850	1.000000	-0.336937
col5	-0.133423	-0.159177	0.207259	-0.336937	1.000000

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to create basic plots

Description

Plotting is a huge topic with many options and variations, but the most foundational types of plots are a line plot and a scatterplot. How can we create those?

Related tasks:

- How to add details to a plot
- How to create a histogram
- How to create a box (and whisker) plot
- How to change axes, ticks, and scale in a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in Python using Matplotlib

We will create some fake data using Python lists, for simplicity. But everything we show below works also if your data is in columns of a DataFrame, such as `df['age']`.

```
patient_id     = [   0,   1,   2,   3,   4,   5,   6,   7,   8,   9 ]
patient_height = [  60,  64,  64,  65,  66,  66,  70,  72,  72,  76 ]
patient_weight = [ 141, 182, 169, 204, 138, 198, 180, 175, 244, 196 ]
```

The conventional way to import matplotlib in Python is as follows.

```
import matplotlib.pyplot as plt
```

Make a line plot by giving the x and y data values in separate lists (or in this case pandas Series). This line plot is very jagged just because the data was random.

```
plt.plot( patient_id, patient_height )        # create plot
plt.show()                                    # display plot
```

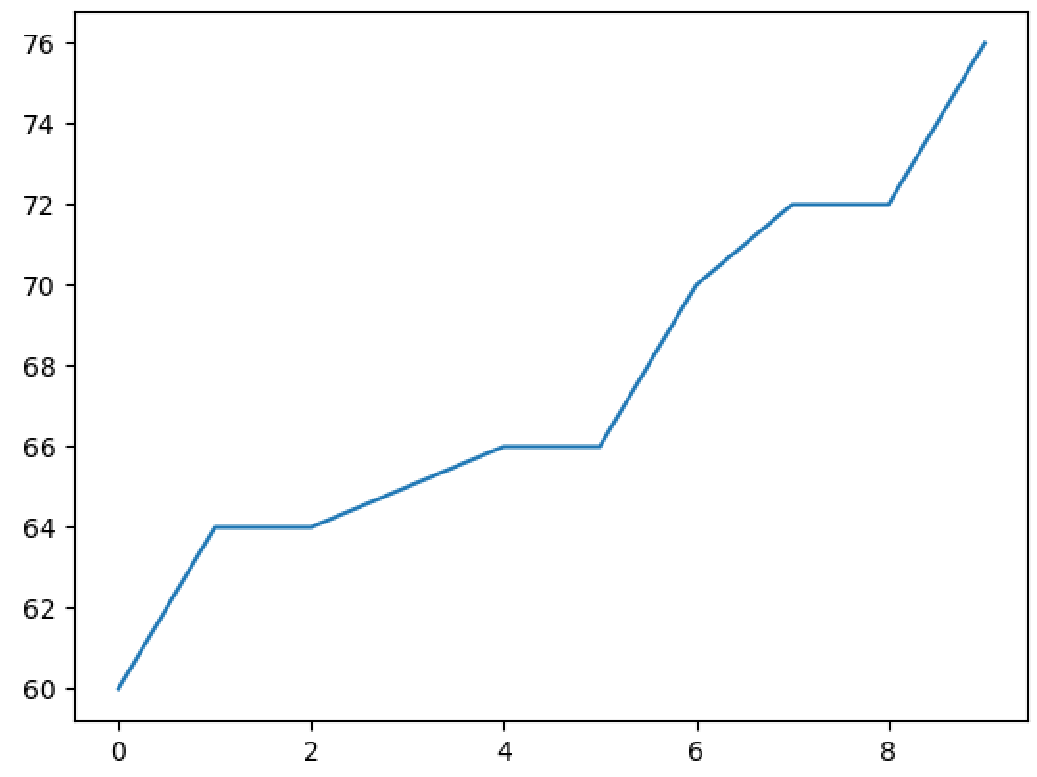

Figure 1: png

You can make a scatterplot as follows.

```
plt.scatter( patient_height, patient_weight )  # create plot
plt.show()                                     # display plot
```

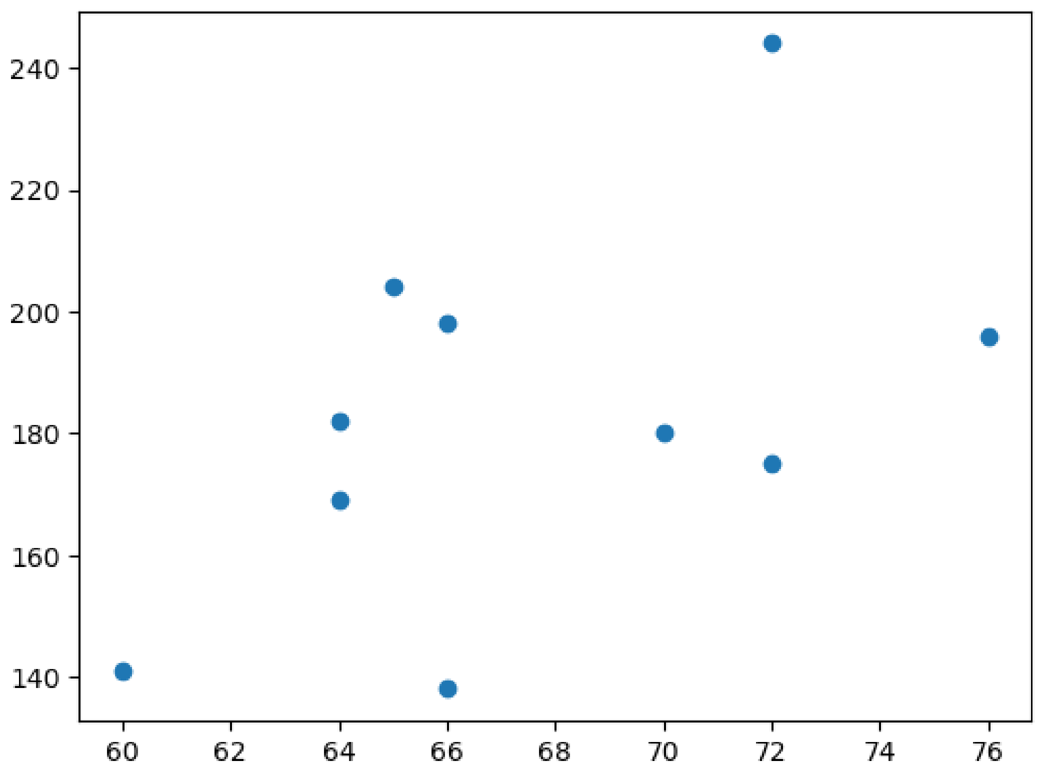

Figure 2: png

If your data is already in a pandas DataFrame, there are shortcuts:

```
# Plot all columns:
df.plot()
plt.show()

# Plot all columns in separate subplots:
df.plot( subplots = True )
plt.show()

# Plot one column:
df['column'].plot()
plt.show()

# Plot specific columns:
df.plot( x='col name', y='other col name' )
plt.show()
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to add details to a plot

Description

After making a plot, we might want to add axis labels, a title, gridlines, or text. Plotting packages provide tons of tools for this sort of thing. What are some of the essentials?

Related tasks:

- How to create basic plots
- How to create a histogram
- How to create a box (and whisker) plot
- How to change axes, ticks, and scale in a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in Python using Matplotlib

We will create some fake data using Python lists, for simplicity. But everything we show below works also if your data is in columns of a DataFrame, such as `df['age']`.

```
patient_height = [  60,  64,  64,  65,  66,  66,  70,  72,  72,  76 ]
patient_weight = [ 141, 182, 169, 204, 138, 198, 180, 175, 244, 196 ]
```

The conventional way to import matplotlib in Python is as follows.

```
import matplotlib.pyplot as plt
```

The following code creates a plot with many details added, but each is independent of the others, so you can take just the bit of code that you need.

```
plt.scatter( patient_height, patient_weight )
plt.xlabel( 'This is the x axis label.' )
plt.ylabel( 'This is the y axis label.' )
plt.title( 'This is the title.' )
plt.grid()                                      # Turns on gridlines
plt.text( 70, 200, 'Text at (70,200)' )         # Text method 1
plt.annotate( 'Text at (60,150)', (60,150) )    # Text method 2
plt.annotate( 'Text with arrow', xytext=(60,225), xy=(72,244),
              arrowprops={'color':'red'} )      # Text with arrow
plt.show()
```

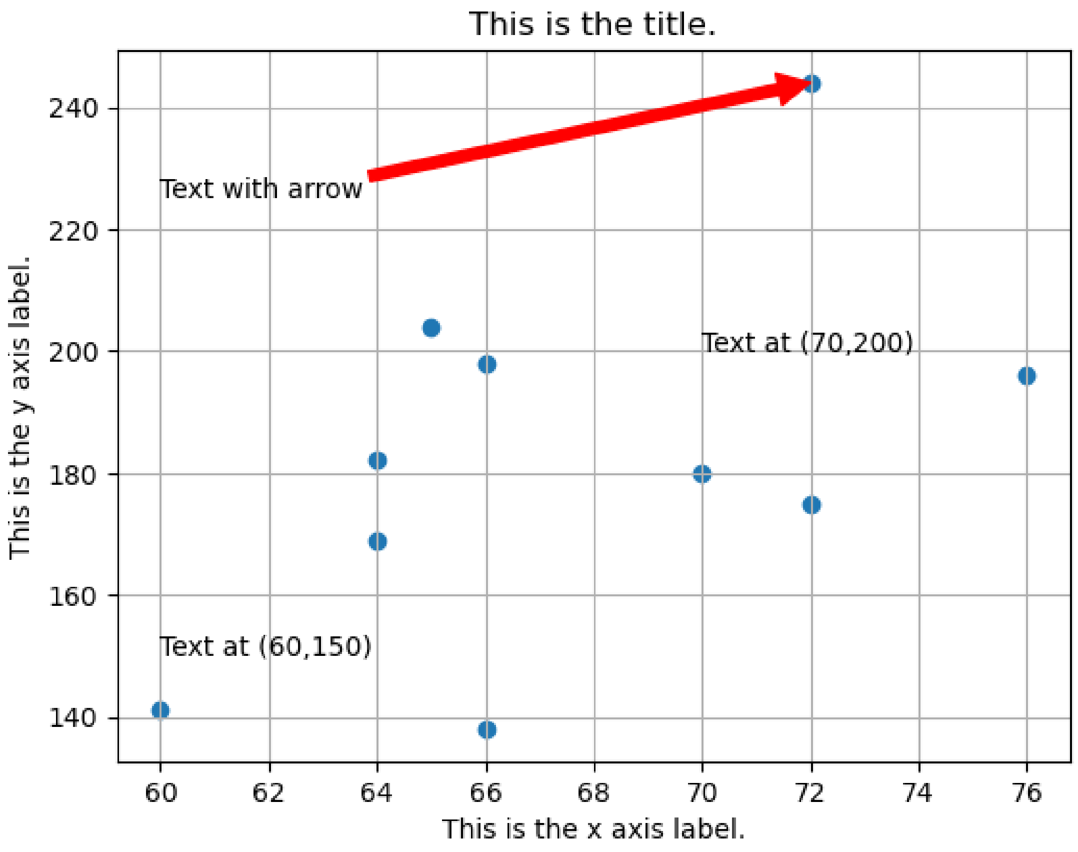

Figure 3: png

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to change axes, ticks, and scale in a plot

Description

The mathematical markings and measurements in a plot can make a big difference on its readability and usefulness. These include the range of each axis, which points on that axis are marked with tick marks, and whether the axes use linear or logarithmic scaling. How can we customize these options?

Related tasks:

- How to create basic plots
- How to create a histogram
- How to create a box (and whisker) plot
- How to add details to a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in Python using Matplotlib

We will create some fake data using Python lists, for simplicity. But everything we show below works also if your data is in columns of a DataFrame, such as `df['age']`.

```
patient_id     = [   0,   1,   2,   3,   4,   5,   6,   7,   8,   9 ]
patient_height = [  60,  64,  64,  65,  66,  66,  70,  72,  72,  76 ]
patient_weight = [ 141, 182, 169, 204, 138, 198, 180, 175, 244, 196 ]
```

The conventional way to import matplotlib in Python is as follows.

```
import matplotlib.pyplot as plt
```

You can change the plot range and where tick marks are shown, in either the x or y directions (or both) as follows.

```
plt.scatter( patient_height, patient_weight )
plt.xlim( [ 0, 80 ] )          # Plot from x=0 to x=80.
plt.ylim( [ 0, 250 ] )         # Plot from y=0 to y=250.
plt.xticks( range(0,80,10) )   # Put x axis ticks every 10 units.
plt.yticks( range(0,250,50) )  # Y ticks every 50.  You can provide any list.
plt.show()
```

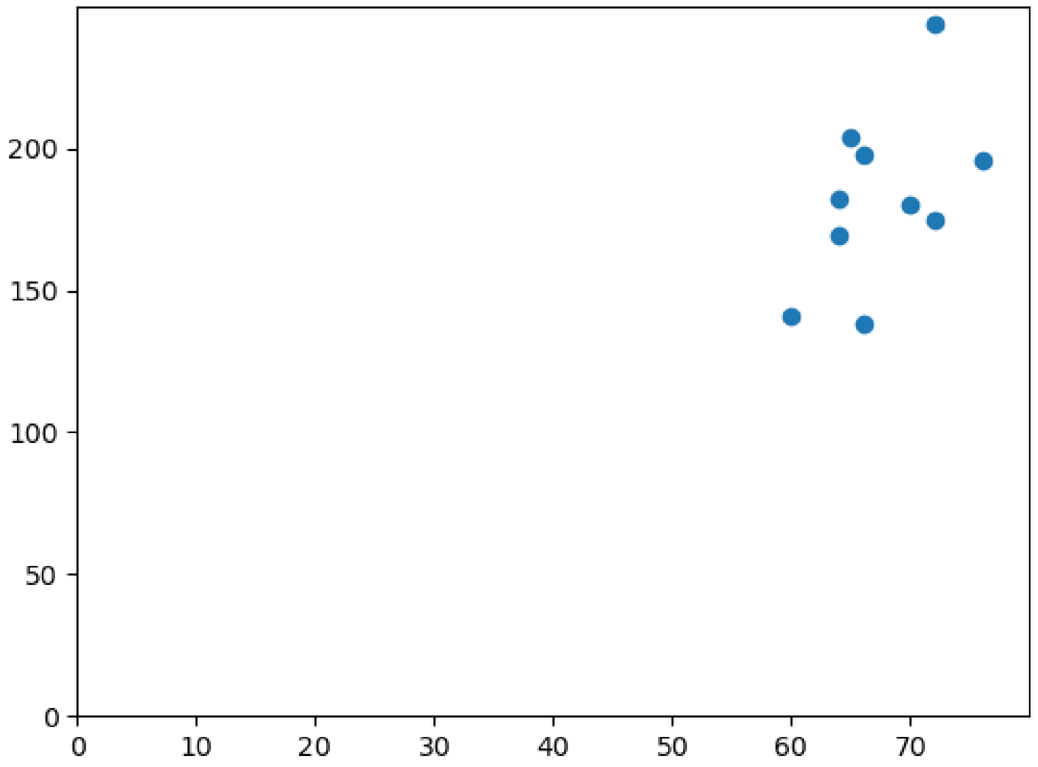

Figure 4: png

If you need either axis to be logarithmically scaled, it takes just one line of code.

```
import numpy as np   # Just using this to make random data.
plt.scatter( np.random.rand(100), np.random.rand(100) )
plt.xscale( 'log' )  # You can include one of these two
plt.yscale( 'log' )  # lines, or both, or neither.
plt.show()
```

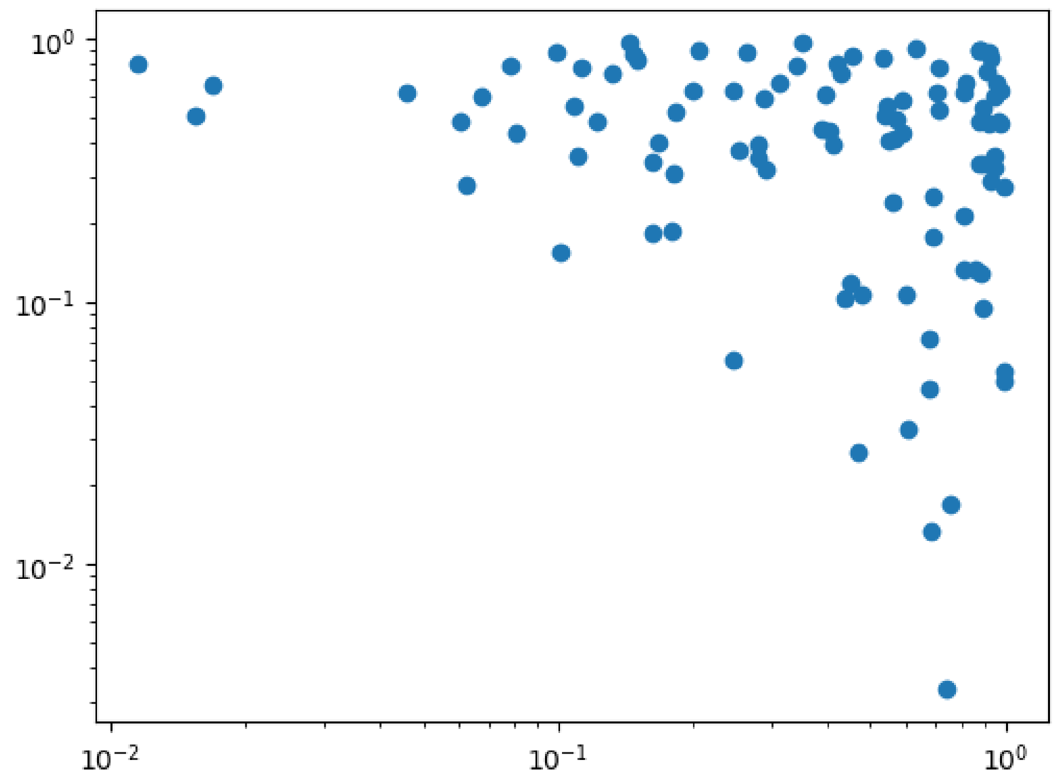

Figure 5: png

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to create a histogram

Description

A histogram is a very common and useful data visualization. It displays an approximation of the distribution in single series of data points (one variable) by grouping the data into bins, each bin draw as a vertical bar. How can we create such a visualization?

Related tasks:

- How to create basic plots
- How to create a box (and whisker) plot
- How to add details to a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in Python using Matplotlib

We will create some random data using NumPy, but that's just for demonstration purposes. You can apply the answer below to any data, even if it's stored just in plain Python lists.

```
import numpy as np
data = np.random.normal( size=1000 )
```

The conventional way to import matplotlib in Python is as follows.

```
import matplotlib.pyplot as plt
```

To create a histogram with 10 bins (the default):

```
plt.hist( data )  # or plt.hist( bins=20 ), or any number
plt.show()
```

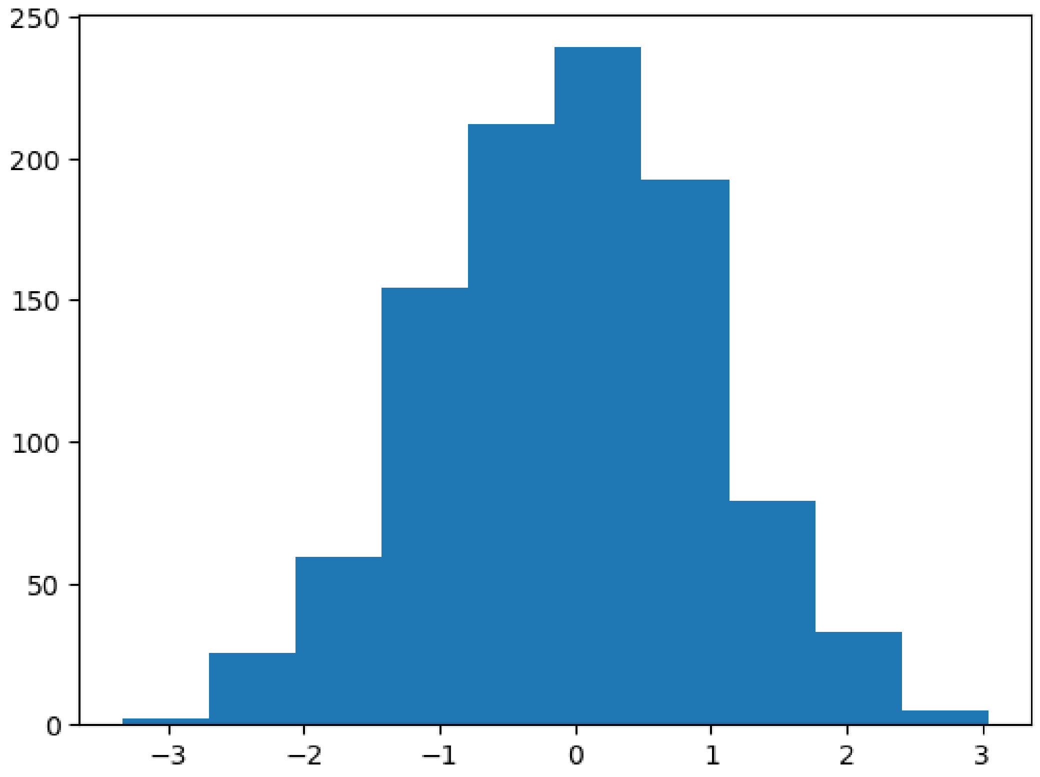

Figure 6: png

The y axis in a histogram is frequency, or number of occurrences. You can change it to probabilities instead.

```
plt.hist( data, density=True )
plt.show()
```

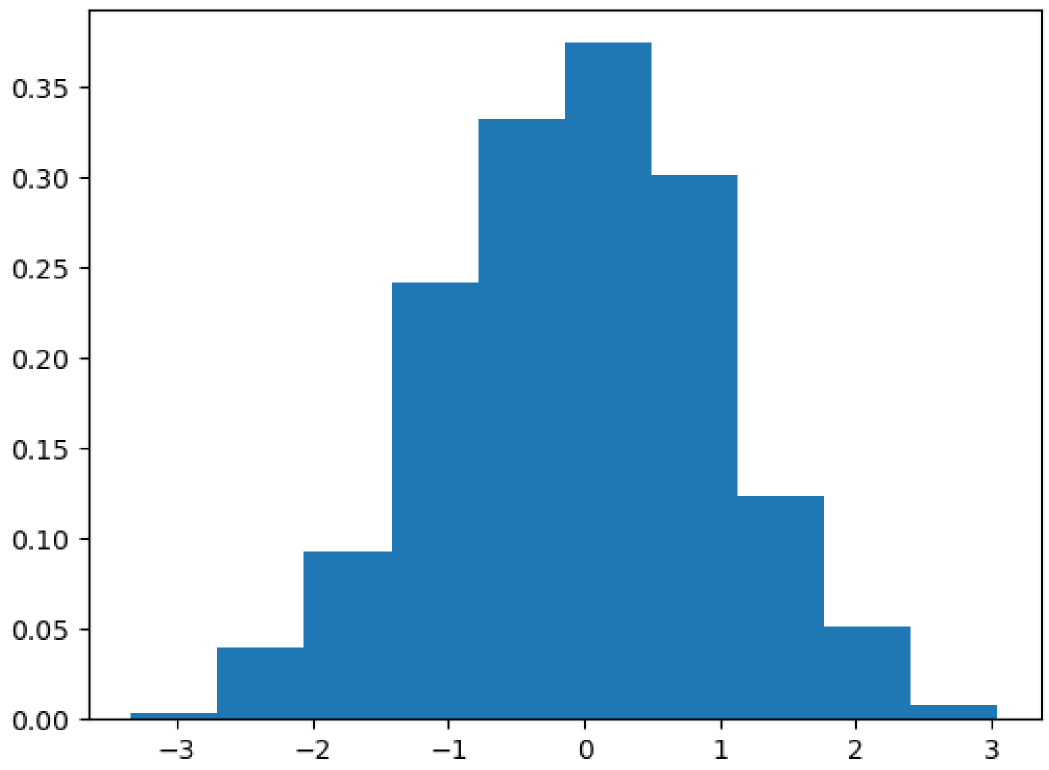

Figure 7: png

You can also choose your own bin boundaries.

```
plt.hist( data, bins=range(-10,10,1) )
plt.show()
```

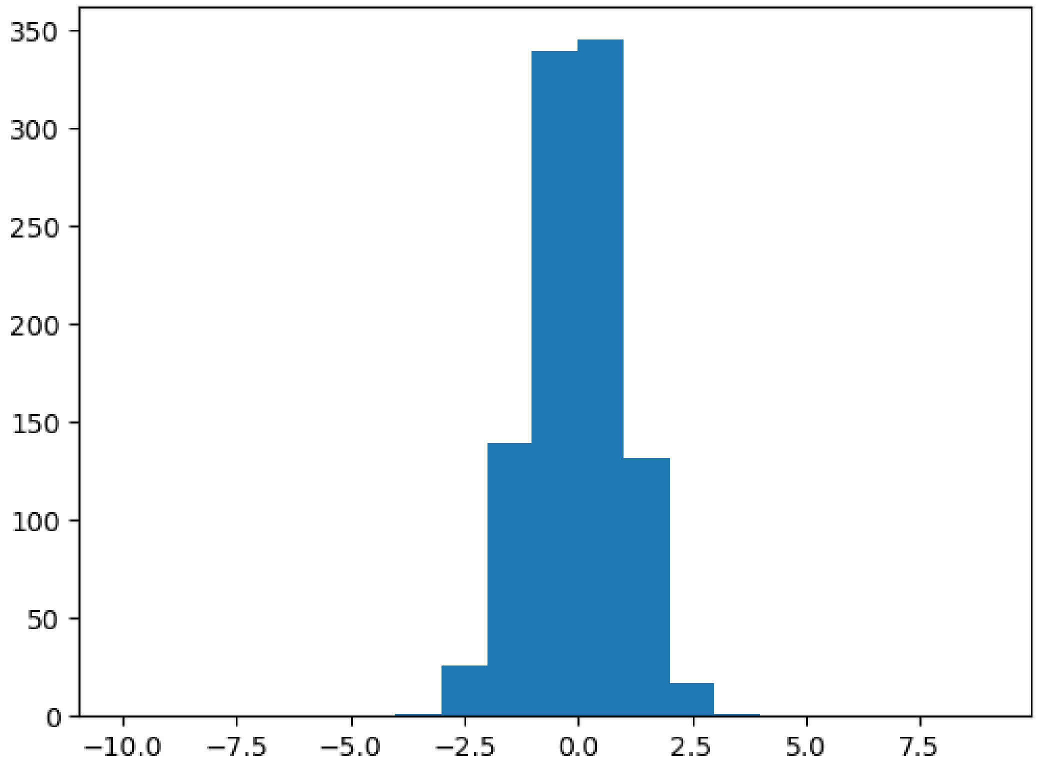

Figure 8: png

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to create a box (and whisker) plot

Description

A box plot, or a box and whisker plot, shows the quartiles of a single variable from a dataset (one of which is the median) and may also show the outliers. It is a simplified way to see the distribution of a variable. Sometimes multiple box plots (one for each of several variables) are shown side-by-side on a plot, to compare the variables. How can we create such graphs?

Related tasks:

- How to create basic plots
- How to add details to a plot
- How to create a histogram
- How to change axes, ticks, and scale in a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in Python using Matplotlib

We will create some fake data using Python lists, for simplicity. But everything we show below works also if your data is in columns of a DataFrame, such as `df['age']`.

```
patient_id     = [   0,   1,   2,   3,   4,   5,   6,   7,   8,   9 ]
patient_height = [  60,  64,  64,  65,  66,  66,  70,  72,  72,  76 ]
patient_weight = [ 141, 182, 169, 204, 138, 198, 180, 175, 244, 196 ]
```

The conventional way to import matplotlib in Python is as follows.

```
import matplotlib.pyplot as plt
```

To create a box-and-whisker plot, sometimes called just a box plot requires just one line of code, plus one to show the plot.

```
plt.boxplot( patient_height )
plt.show()
```

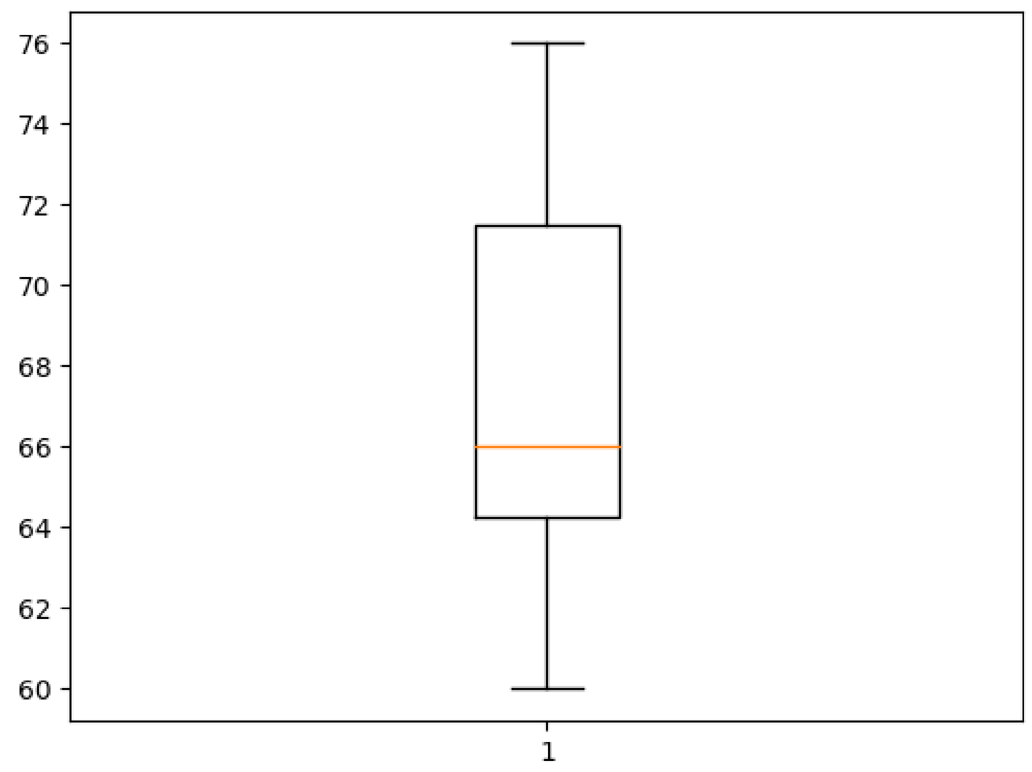

Figure 9: png

You can show more than one variable's box plot side-by-side by forming a list of the data.

```
plt.boxplot( [ patient_height, patient_weight ] )
plt.show()
```

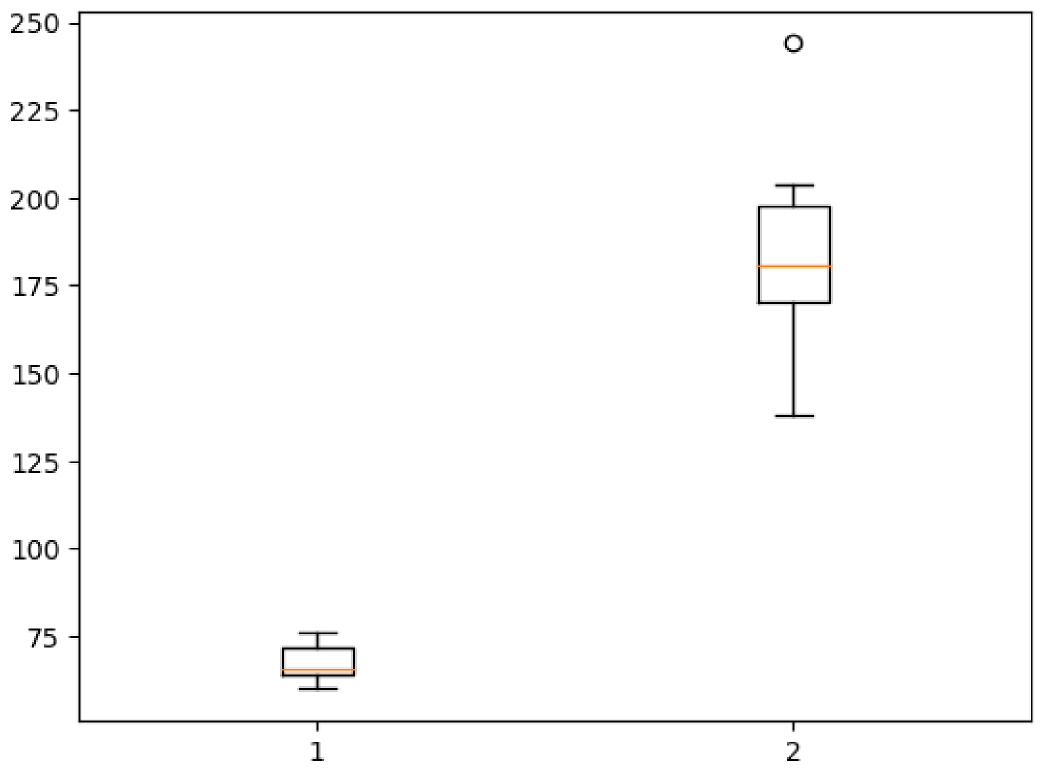

Figure 10: png

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to create a QQ-plot

Description

We often want to know whether a set of data is normally distributed, so that we can deduce what inference tests are appropriate to conduct. If we have a set of data and want to figure out if it comes from a population that follows a normal distribution, one tool that can help is a QQ plot. How do we make and interpret one?

Related tasks:

- How to test data for normality with Pearson's chi-squared test (on website)
- How to test data for normality with the D'Agostino-Pearson test (on website)
- How to test data for normality with the Jarque-Bera test (on website)

Solution in Python using SciPy

We're going to use some fake data here by generating random numbers, but you can replace our fake data with your real data in the code below.

```
# Replace this with your data, such as a variable or column in a DataFrame
import numpy as np
values = np.random.normal(0, 1, 50)  # 50 random values
```

If the data is normally distributed, then we expect that the QQ plot will show the observed values (blue dots) falling very clsoe to the red line (the quantiles for the normal distribution).

```
from scipy import stats
import matplotlib.pyplot as plt

stats.probplot(values, dist="norm", plot=plt)
plt.show()
```

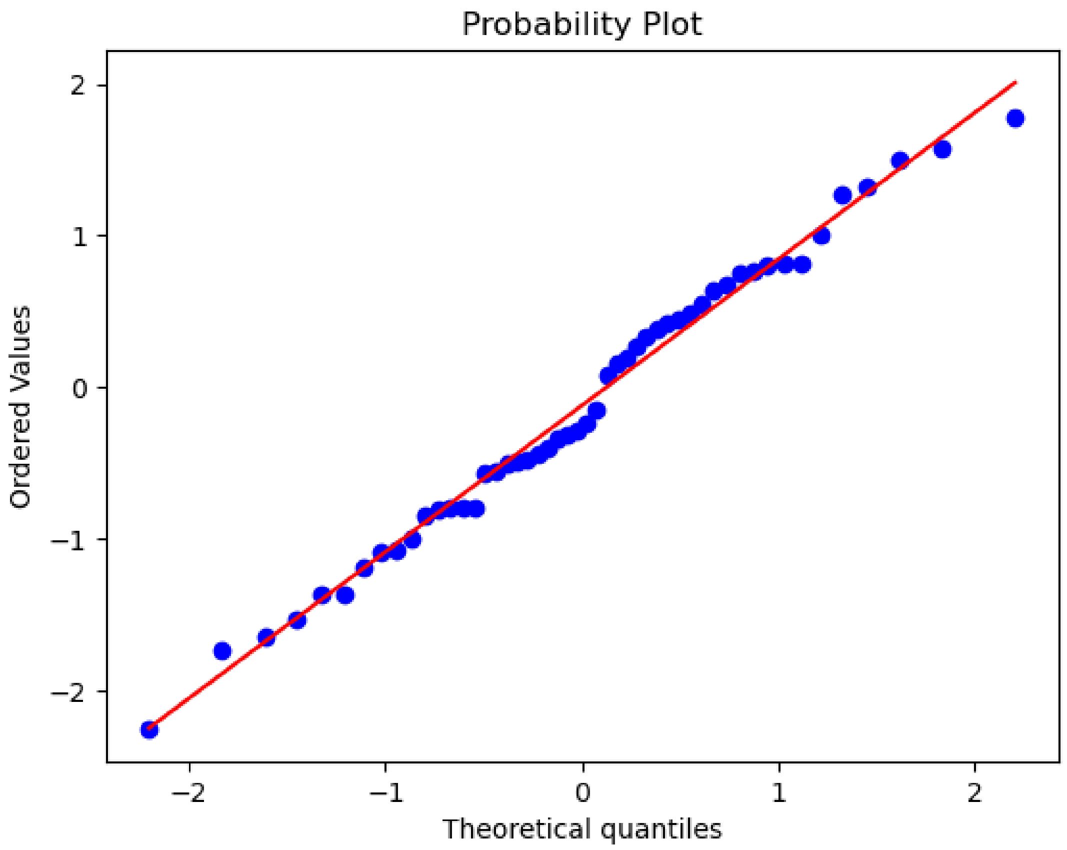

Figure 11: png

Our observed values fall pretty close to the reference line. In this case, we expected that, because we created fake data that was normally distributed. But for real data, it may not stay so close to the red line.

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.