

# Programming for Essential Digital Skills, Part 2

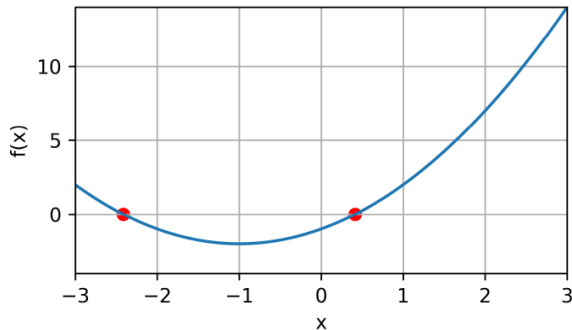
Pieter Kler

2024

## Chapter 8 - Mathematics and plotting

# Root finding

Consider  $f(x) = x^2 + 2x - 1$ . A **root**  $x$  of the function  $f$  is a point that satisfies  $f(x) = 0$ .



# SciPy: Scientific Computing with Python

## Subpackages

SciPy is organized into subpackages covering different scientific computing domains. These are summarized in the following table:

Subpackage	Description
<code>cluster</code>	Clustering algorithms
<code>constants</code>	Physical and mathematical constants
<code>fft</code>	Discrete Fourier transforms
<code>fftpack</code>	Fast Fourier Transform routines (legacy)
<code>integrate</code>	Integration and ordinary differential equation solvers
<code>interpolate</code>	Interpolation and smoothing splines
<code>io</code>	Input and Output
<code>linalg</code>	Linear algebra
<code>ndimage</code>	N-dimensional image processing
<code>odr</code>	Orthogonal distance regression
<code>optimize</code>	Optimization and root-finding routines

Solving the equation  $f(x) = 0$  using `fsolve()`

## Solving the equation $f(x) = 0$ using `fsolve()`

```
import scipy.optimize as optimize

def f(x):
    return x**2 + 2*x - 1

guess = 3
f_zero = optimize.fsolve(f,guess)[0]

print("A root of the function f is given by", f_zero)
```

A root of the function f is given by 0.41421356237309503

## Solving the equation $f(x) = 3$

Suppose we want to solve  $f(x) = 3$ . How to do this with root finding?

## Solving the equation $f(x) = 3$

Suppose we want to solve  $f(x) = 3$ . How to do this with root finding?

- If we define  $g(x) = f(x) - 3$ , then  $g(x) = 0$  if and only if  $f(x) = 3$ .



## Solving the equation $f(x) = 3$

Suppose we want to solve  $f(x) = 3$ . How to do this with root finding?

- If we define  $g(x) = f(x) - 3$ , then  $g(x) = 0$  if and only if  $f(x) = 3$ .

```
def g(x):  
    return f(x) - 3  
  
guess = 4  
f_zero = optimize.fsolve(g,guess)[0]  
  
print("A number x satisfying f(x) = 3, is given by", f_zero)
```

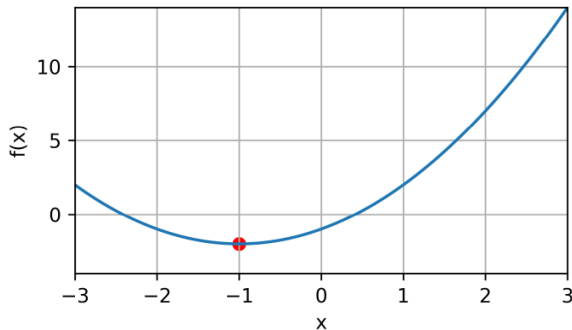
A number x satisfying  $f(x) = 3$ , is given by 1.2360679774998171

## Solving the equation $f(x) = c$

```
def solve_eq(f,c,guess):  
    # This function returns the solution to  $f(x) = c$  using  
    # fsolve() on the function  $g(x) = f(x) - c$   
  
    def g(x):  
        return f(x) - c  
  
    x = optimize.fsolve(g,guess)[0]  
    return x
```

## Minimizing a function $f$

Consider  $f(x) = x^2 + 2x - 1$ . **Minimum of  $f$**  is a point  $x$  for which  $f(x)$  is smallest.



## Computing a minimum of $f$ using `fmin()`

```
import scipy.optimize as optimize
```

```
def f(x):  
    return x**2 + 2*x - 1
```

```
guess = 1  
minimum = optimize.fmin(f,guess)
```

Optimization terminated successfully.

Current function value: -2.000000

Iterations: 19

Function evaluations: 38

```
print('The minimum of the function f is attained at x = ', minimum)
```

The minimum of the function  $f$  is attained at  $x = [-1.]$

## Computing a minimum of $f$ using `fmin()`

```
import scipy.optimize as optimize

def f(x):
    return x**2 + 2*x - 1

guess = 1
minimum = optimize.fmin(f,guess,disp=False)[0]

print('The minimum of the function f is attained at x = ', minimum)
```

The minimum of the function f is attained at x = -1.00000000000000018

## Computing a minimum of $f$ using `fmin()`

```
import scipy.optimize as optimize

def f(x):
    return x**2 + 2*x - 1

guess = 1
minimum = optimize.fmin(f,guess,disp=False)[0]

print('The minimum of the function f is attained at x = ', minimum)
```

The minimum of the function  $f$  is attained at  $x = -1.00000000000000018$

Note: `fmin()` might return a “local” minimum, which is not the true minimum of the function (Classroom Exercise 1).

# Matplotlib: Data visualization

Matplotlib is a package that can be used for data visualization

- For this we use the `matplotlib.pyplot` (sub)package ...
- ... which we usually import under the name `plt`

# How are functions plotted in Python?

- 1 Create a vector of  $x$ -values, e.g.,

$$x = [-3, -2, -1, 0, 1, 2, 3].$$

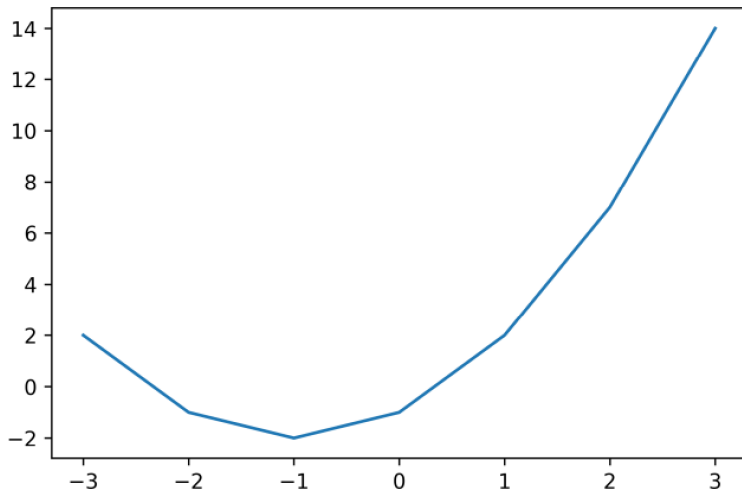
- 2 Compute the function values

$$[f(-3), f(-2), f(-1), f(0), f(1), f(2), f(3)] = [2, -1, -2, -1, 2, 7, 14].$$

- 3 Draw the points  $(x_i, f(x_i))$  and connect them with line segments.



## Resulting Python plot



## Plotting a “smooth” line

Increase the number of points in  $x$  to get a smoother line using `np.linspace()`.

- Command `np.linspace(a,b,k)` plots  $k$  evenly spaced points in interval  $[a, b]$ .

## Plotting a “smooth” line

Increase the number of points in  $x$  to get a smoother line using `np.linspace()`.

- Command `np.linspace(a,b,k)` plots  $k$  evenly spaced points in interval  $[a, b]$ .

```
import numpy as np
```

```
a = 0
```

```
b = 1
```

```
k = 11
```

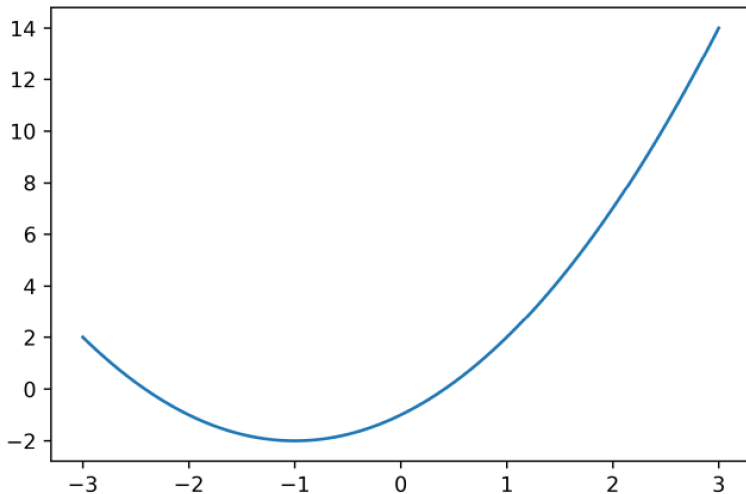
```
x = np.linspace(a,b,k)
```

```
print(x)
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

## Resulting “smoothed” Python plot

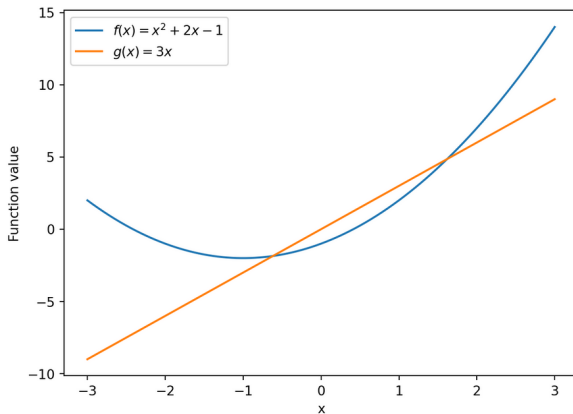
Using `x = np.linspace(-3,3,600)`



## Adding legend to plot

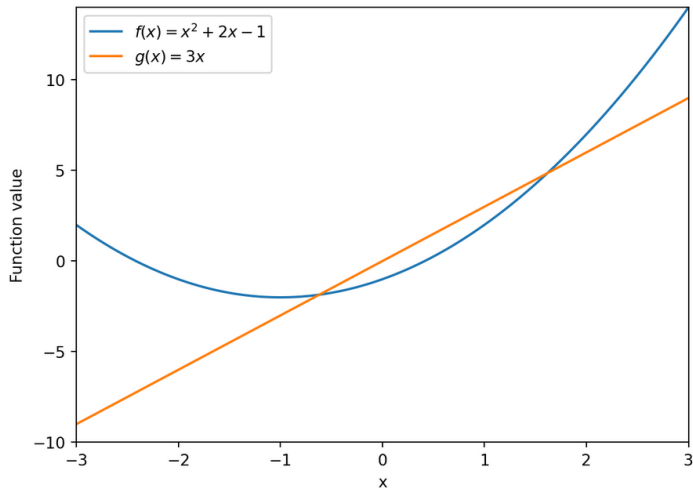
Use label-argument in `plt.plot()` in combination with `plt.legend()` at the end ...

- ... and `plt.xlabel('x')` and `plt.ylabel('Function value')` for axis labels.



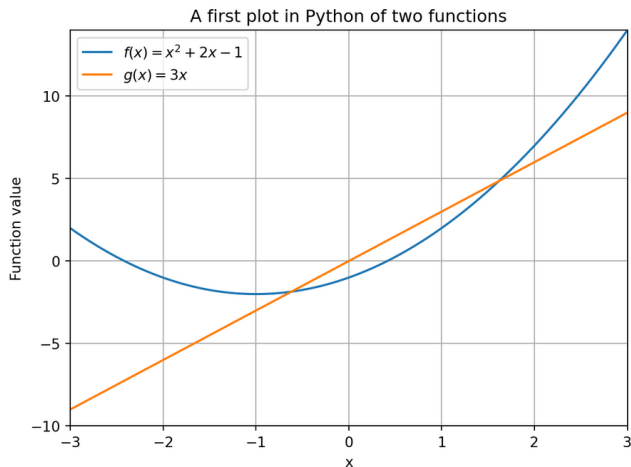
## Fixing axes ranges

Use `plt.xlim(-3,3)` and `plt.ylim(-10,14)` to fix range of horizontal/vertical axis, resp.



## Adding title and grid

- Use `plt.title('A first plot of two functions')` to add title
- Use `plt.grid()` to add grid.



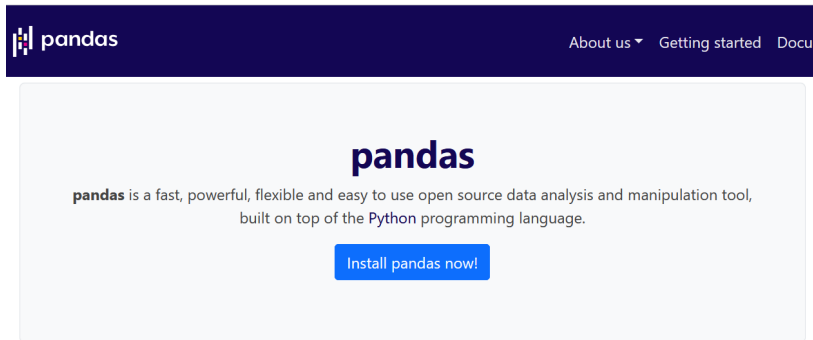
## Classroom Exercise 1

Consider the function  $f(x) = \frac{9}{10}x^4 - 3x^3 - \frac{7}{2}x^2 + 12x + 3$ .

- Plot this function with horizontal axis range  $[-6, 6]$ , and vertical axis range  $[-15, 15]$ .
- Find four roots of this function with `fsolve()` by trying out different initial guesses.
- Find a minimum of this function with `fmin()` by using initial guesses  $-1$  and  $2$ . Are both solutions actual minima of the function?



## Chapter 9 - Data handling with Pandas



Import the package under alias `pd`

```
import pandas as pd
```

# Input data

Data can come from many sources. We look at two possibilities:

- Python dictionary (data contained in Python script)
- Matrix, i.e., list of lists (data contained in Python script)
- Comma-separated values (CSV) file (data loaded into Python from another file)

## Data in dictionary

```
# Dictionary with data
dataset = {
    'name' : ["Aiden", "Bella", "Carlos", "Dalia", "Elena", "Farhan"],
    'height (cm)' : [185, 155, 190, 185, 160, 170],
    'weight (kg)' : [80, 60, 100, 85, 62, 75],
    'age (years)' : [23, 23, 23, 21, 19, 25],
    'dietary preference' : ['Veggie', 'Veggie', 'None', 'None', 'Vegan', 'None']
}
```

## Pandas data frame (created from dictionary)

We load data into a data frame using `DataFrame()` whose input argument is our dictionary.

```
data_frame = pd.DataFrame(dataset)
```

```
print(data_frame)
```

	name	height (cm)	weight (kg)	age (years)	dietary preference
0	Aiden	185	80	23	Veggie
1	Bella	155	60	23	Veggie
2	Carlos	190	100	23	None
3	Dalia	185	85	21	None
4	Elena	160	62	19	Vegan
5	Farhan	170	75	25	None

# Frame info

Frame columns (dictionary keys)

	name	height (cm)	weight (kg)	age (years)	dietary preference
0	Aiden	185	80	23	Veggie
1	Bella	155	60	23	Veggie
2	Carlos	190	100	23	None
3	Dalia	185	85	21	None
4	Elena	160	62	19	Vegan
5	Farhan	170	75	25	None

Frame indices

Frame column data (dictionary value)

# Accessing data from Pandas frame

Suppose we have a frame named `frame_name`.

Command	Output
<code>frame_name.head(k)</code>	First $k$ rows of frame
<code>frame_name.tail(k)</code>	Last $k$ rows of frame
<code>frame_name.loc[i]</code>	Row with index $i$
<code>frame_name.loc[i,col_name]</code>	Element on row $i$ in column <code>col_name</code>
<code>frame_name.loc[:,col_name]</code> or <code>frame_name[col_name]</code>	Column <code>col_name</code>

## Accessing blocks of the frame

With `frame_name.loc[[i1,i2,...,iq],[col_1,...,col_r]]`:

- Block formed by row indices  $i_1, \dots, i_q$  and columns with name  $col_1, \dots, col_r$

```
# Extract block with rows 2,4 and columns name, height and age.  
x = data_frame.loc[[2,4],['name','height (cm)', 'age (years)']]  
print(x)
```

	name	height (cm)	age (years)
2	Carlos	190	23
4	Elena	160	19

With `frame_name.loc[[i1,i2,...,iq]]`:

- Block formed by row indices  $i_1, \dots, i_q$  and all columns.



## Accessing rows using Boolean list

For list  $x$  containing values True or False:

- Access True-rows with `frame_name.loc[x]`.

```
x = [False, False, True, False, True, False]
print(data_frame.loc[x])
```

	name	height (cm)	weight (kg)	age (years)	dietary preference
2	Carlos	190	100	23	None
4	Elena	160	62	19	Vegan

## Accessing rows based on conditional statement

Suppose we only want the people whose dietary preference is 'None'.

- Conditional statement `data_frame.loc[:, 'dietary preference'] == 'None'`

```
x = data_frame.loc[:, 'dietary preference'] == 'None'  
print(x)
```

```
0    False
```

```
1    False
```

```
2     True
```

```
3     True
```

```
4    False
```

```
5     True
```

```
Name: dietary preference, dtype: bool
```

## Accessing rows based on conditional statement (cont'd)

```
x = data_frame.loc[:, 'dietary preference'] == 'None'  
print(data_frame.loc[x])
```

	name	height (cm)	weight (kg)	age (years)	dietary preference
2	Carlos	190	100	23	None
3	Dalia	185	85	21	None
5	Farhan	170	75	25	None

## Editing frame data

```
data = [  
    [2,4,-1,2],  
    [5,1,2,9],  
    [3,7,8,9]  
]  
  
frame = pd.DataFrame(data)  
  
print(frame)
```

	0	1	2	3
0	2	4	-1	2
1	5	1	2	9
2	3	7	8	9

## Editing row names

Row and column names are stored in `frame.index` and `frame.columns`, respectively.

```
print(frame.index)
```

```
RangeIndex(start=0, stop=3, step=1)
```

```
frame.index = ['Row0', 'Row1', 'Row2']  
print(frame)
```

	0	1	2	3
Row0	2	4	-1	2
Row1	5	1	2	9
Row2	3	7	8	9

## Editing column names

```
frame.columns = ['Col0', 'Col1', 'Col2', 'Col3']  
  
print(frame)
```

	Col0	Col1	Col2	Col3
Row0	2	4	-1	2
Row1	5	1	2	9
Row2	3	7	8	9

## Editing entries

Editing entries can be done with `frame.loc[row_name,col_name] = new_value`

```
# Edit entry on row 1, column 2
frame.loc['Row1','Col2'] = 10

print(frame)
```

	Col0	Col1	Col2	Col3
Row0	2	4	-1	2
Row1	5	1	10	9
Row2	3	7	8	9

## Editing whole row

Replace row with list y: `frame.loc[row_name,:] = y`

```
# Replace row 2
y = [-2,-2,-2,-2]
frame.loc['Row2'] = y #frame.loc['Row2',:] = y also works

print(frame)
```

	Col0	Col1	Col2	Col3
Row0	2	4	-1	2
Row1	5	1	10	9
Row2	-2	-2	-2	-2



## Editing whole column

Replace column with list y: `frame.loc[:,col_name] = y`

```
# Replace column 2
y = [-1,-1,-1]
frame.loc[:, 'Col2'] = y

print(frame)
```

	Col0	Col1	Col2	Col3
Row0	2	4	-1	2
Row1	5	1	-1	9
Row2	-2	-2	-1	-2

## Editing whole column according to a mathematical function

Suppose we want to square every number in the column 'Col1'.

```
def f(x):  
    return x**2
```

Use `apply()` function and overwrite entries in 'Col1' (don't use `frame.loc['Col1']!`)

```
frame['Col1'] = frame['Col1'].apply(f)  
  
print(frame)
```

	Col0	Col1	Col2	Col3
Row0	2	16	-1	2
Row1	5	1	-1	9
Row2	-2	4	-1	-2

## Adding new row

New rows appear at the bottom of the frame.

```
# Add a row  
frame.loc['New row'] = [5,5,3,1]  
print(frame)
```

	Col0	Col1	Col2	Col3
Row0	2	16	-1	2
Row1	5	1	-1	9
Row2	-2	4	-1	-2
New row	5	5	3	1

## Adding new column

New columns appear at the right of the frame.

```
frame.loc[:, 'New column'] = [1,1,1,1]  
print(frame)
```

	Col0	Col1	Col2	Col3	New column
Row0	2	16	-1	2	1
Row1	5	1	-1	9	1
Row2	-2	4	-1	-2	1
New row	5	5	3	1	1

## Inserting a new column

Can also insert column at specified location using `frame.insert()`

- Takes as input insertion position, column name and column data.

```
# Insert column with name 'New column' and data [10,10,10,10] at position 2.  
frame.insert(2,'Inserted column', [10,10,10,10])  
  
print(frame)
```

	Col0	Col1	Inserted column	Col2	Col3	New column
Row0	2	16	10	-1	2	1
Row1	5	1	10	-1	9	1
Row2	-2	4	10	-1	-2	1
New row	5	5	10	3	1	1

## Computing statistics of column data

Can compute properties of data like minimum, maximum, mean, etc.

```
print(frame)
```

	Col0	Col1	Inserted column	Col2	Col3	New column
Row0	2	16	10	-1	2	1
Row1	5	1	10	-1	9	1
Row2	-2	4	10	-1	-2	1
New row	5	5	10	3	1	1

```
# Minimum of the first column
```

```
min_col1 = frame.loc[:, 'Col1'].min() #Use .max()/mean() for maximum/mean  
print(min_col1)
```

## Classroom exercise

```
data = [  
    [2,4,-1,2],  
    [5,1,2,9],  
    [3,7,8,9]  
]  
  
frame = pd.DataFrame(data)  
frame.index = ['Row0','Row1','Row2']  
frame.columns = ['Col0','Col1','Col2','Col3']
```

Create the following Pandas data frame that has an extra row with the maximum per column

	Col0	Col1	Col2	Col3
Row0	2	4	-1	2
Row1	5	1	2	9
Row2	3	7	8	9
Maximum	5	7	8	9

# Importing data into Python

Data from, e.g., comma-separated values (CSV) file can be imported into Python using `read_csv()`.

```
csv_to_frame = pd.read_csv('dataset.csv')  
  
print(csv_to_frame)
```

	name	height (cm)	weight (kg)	age (years)	dietary preference
0	Aiden	185	80	23	Veggie
1	Bella	155	60	23	Veggie
2	Carlos	190	100	23	None
3	Dalia	185	85	21	None
4	Elena	160	62	19	Vegan
5	Farhan	170	75	25	None
6	Geert	178	80	25	Veggie



## Data header

Python interprets first line of .csv file as header with column names for the Pandas frame.

- No header present? Set `header=None` as an additional argument in `read_csv()`.

```
csv_to_frame = pd.read_csv('dataset.csv', header=None)
print(csv_to_frame)
```

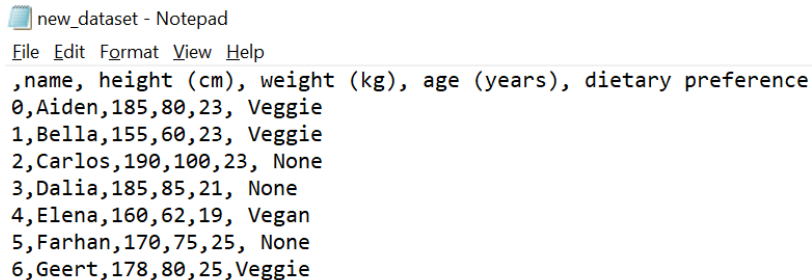
	0	1	2	3	4
0	name	height (cm)	weight (kg)	age (years)	dietary preference
1	Aiden	185	80	23	Veggie
2	Bella	155	60	23	Veggie
3	Carlos	190	100	23	None
4	Dalia	185	85	21	None
5	Elena	160	62	19	Vegan
6	Farhan	170	75	25	None
7	Geert	178	80	25	Veggie

# Exporting data out of Python

Can export Pandas frame to a .csv file using `to_csv()`.

```
frame.to_csv('new_dataset.csv')
```

This creates new file in same folder as Python script with the given name.



The screenshot shows a Notepad window titled "new\_dataset - Notepad". The menu bar includes File, Edit, Format, View, and Help. The text content is a CSV file with a header row and seven data rows. Each data row starts with an index from 0 to 6, followed by a name, height in cm, weight in kg, age in years, and dietary preference. The data is as follows:

	name	height (cm)	weight (kg)	age (years)	dietary preference
0	Aiden	185	80	23	Veggie
1	Bella	155	60	23	Veggie
2	Carlos	190	100	23	None
3	Dalia	185	85	21	None
4	Elena	160	62	19	Vegan
5	Farhan	170	75	25	None
6	Geert	178	80	25	Veggie

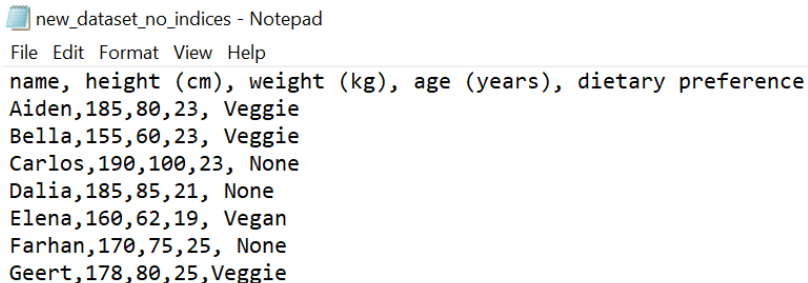
Figure 1: Exported data in .csv file (with row indices)

## Suppressing row indices in exported file

By default, Python includes the row indices in the exported .csv file.

- Suppress row indices in exported file with `index=False` in `to_csv()`.

```
frame.to_csv('new_dataset.csv', index=False)
```



The screenshot shows a Notepad window titled "new\_dataset\_no\_indices - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The text content is a CSV file with the following data:

name	height (cm)	weight (kg)	age (years)	dietary preference
Aiden	185	80	23	Veggie
Bella	155	60	23	Veggie
Carlos	190	100	23	None
Dalia	185	85	21	None
Elena	160	62	19	Vegan
Farhan	170	75	25	None
Geert	178	80	25	Veggie

Figure 2: Exported data in .csv file (without row indices)