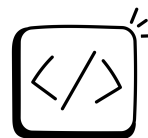
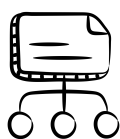


## User guide: Synthetic data tool

Generating low fidelity synthetic data with BIT's Python tool

Easier and safer synthetic data in 4 steps



Many tools exist to generate synthetic data. This tool's specific benefit is that it *only* allows you to generate low fidelity data: the safer, lower risk form of synthetic data.

And it works out of the box: just point it at your data set and in just **four steps** you're creating.

[Skip ahead to Python step-by-step instructions](#)

# Background

## What is synthetic data?

Synthetic data is a randomly generated version of a data set that follows the structure and some of the patterns found in the original data set. “Each piece of information in the data set is meant to be plausible (e.g., an athlete’s height will usually be between 1.5 and 2.2 metres, and would never be 1 kilometre), but it is chosen randomly from the range of possible values, not by pointing to any original individual in the data set. Data that is generated in this way reveals very little, if anything, about any individual in the original data set, but still represents the data well as a whole.”<sup>1</sup>

## Why use synthetic data?

Data that researchers use often contains potentially identifying and/or sensitive information about individuals. While access to such data can be granted, obtaining it is time consuming, often causing long delays for projects. It’s also sometimes unclear *ahead of time* whether the information that a researcher is interested in will be present in the requested data set, which can lead to further delays and complications.

Synthetic data that preserves some qualities of the original data without reproducing the data corresponding to actual individuals can be a way around these problems. Because synthetic data doesn’t exactly reproduce the records of any individuals, synthetic data doesn’t need to have such stringent access requirements. This means that it can be used for testing analysis code or (in the case of high fidelity synthetic data) for exploratory analysis while a researcher waits for access to the real data to be granted.

## Low-fidelity data preserves privacy, reduces paperwork burdens and speeds up project timelines

Synthetic data comes on a scale from low fidelity to high fidelity, based on how many statistical relationships present in the original data are also present in the synthetic data. [Table 1](#) below illustrates what we mean by low and high fidelity synthetic data. Very high fidelity synthetic data may be subject to access controls as stringent as that of real data as it resembles it so closely; low fidelity synthetic data does not necessarily require this as the relationships preserved may be only the format of the data or the statistical relationships present in individual columns and not the relationships between them.

---

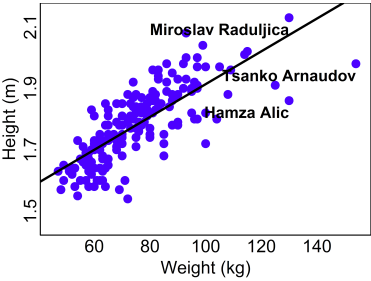
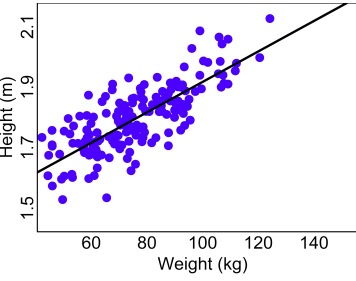
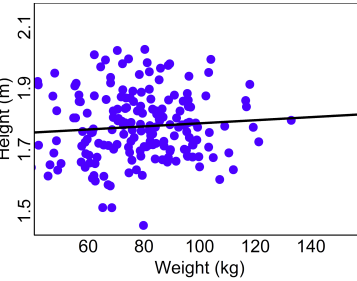
<sup>1</sup> Calcraft *et al.* (2021) ‘Accelerating Public Policy Research with Synthetic Data’ BIT report page 6. Available at: [https://www.adruk.org/fileadmin/uploads/adruk/Documents/Accelerating\\_public\\_policy\\_research\\_with\\_synthetic\\_data\\_December\\_2021.pdf](https://www.adruk.org/fileadmin/uploads/adruk/Documents/Accelerating_public_policy_research_with_synthetic_data_December_2021.pdf).

The BIT synthetic data tool is a tool intended for producing *low fidelity* synthetic data. Because it doesn't preserve the statistical relationships between columns, synthetic data produced by this tool has a low risk of reproducing records resembling real people or being disclosive in other ways. However, it reproduces the structure of the dataset, including erroneous values. This means that it's suitable for use as test datasets that can be released to researchers and analysts so that they can test their analysis code while waiting to be granted access to the real data.

Low fidelity synthetic data might also be used in the early stages of training researchers and analysts on how to use specific data sets. Because the only correct statistics in the synthetic data are the statistical distributions within each column, it won't be very useful if the intent of the training is to uncover what correlations are present in the data. But it should be suitable for training intended to introduce researchers and analysts to the general characteristics and structure of the data set and its quirks.

## Low and high fidelity synthetic data

**Table 1:** At-a-glance comparison of original, high-fidelity and low-fidelity synthetic data.<sup>2</sup>

Original data	High-fidelity synthetic data	Low-fidelity synthetic data
		
<p>Height and weight of every athlete at the 2016 Olympics.</p> <ul style="list-style-type: none"> <li>• <b>Every point corresponds to a real person.</b></li> <li>• <b>Weight increases with height, on average.</b></li> </ul>	<p>Each point is randomly generated from <b>the general relationship between height and weight</b> at the 2016 Olympics.</p> <ul style="list-style-type: none"> <li>• <b>No points correspond to a real person.</b></li> <li>• <b>Weight increases with height, on average.</b></li> </ul>	<p>Each point is randomly generated from <b>the general statistics of height and the general statistics of weight</b> at the 2016 Olympics.</p> <ul style="list-style-type: none"> <li>• <b>No points correspond to a real person.</b></li> <li>• <b>No particular relationship between height and weight.</b></li> </ul>

<sup>2</sup> Simplified from page 8 of Calcraft *et al.* (2021) 'Accelerating Public Policy Research with Synthetic Data' BIT report. Available at: [https://www.adruk.org/fileadmin/uploads/adruk/Documents/Accelerating\\_public\\_policy\\_research\\_with\\_synthetic\\_data\\_December\\_2021.pdf](https://www.adruk.org/fileadmin/uploads/adruk/Documents/Accelerating_public_policy_research_with_synthetic_data_December_2021.pdf).

## BIT's tool makes creating low-fidelity data easier

This tool generates low fidelity synthetic data. This data preserves the structure of the original datafile and generates data for each column based on the statistical distribution of values in that column alone (i.e. univariate statistics only).

This tool removes the statistical relationships between columns. This means that strange combinations of values are possible for individual records. For example, if there are columns for age and date of birth in a data set whose contents should be up to date for the year of 2015, it might contain a record where a five year old was born in 1900. No correlations between values in different columns are intentionally reproduced. This reduces the risk of accidentally reproducing data corresponding to real people.

Synthetic data produced by this tool should be used **only to test analysis code, and never for final analyses**. It only crudely resembles the real data and cannot substitute for it. It can, however, help test whether analysis code works during the early stages of a project while waiting for access to the real data to be granted.




## Low fidelity data still need disclosure processes

**Don't *assume* that undesired features of the original data set won't appear in the output: *check* that they don't!**

While synthetic data generated with this tool will not intentionally reproduce features of the original data that might reveal sensitive or private information, it should not be released without first making sure that these features have not been reproduced **accidentally**. We therefore recommend that some kind of disclosure process should be in place for this data.

### Why is disclosure control needed?

While low-fidelity data reduces risk, it does not eliminate it. Some things to consider before release:

-   **Misclassification** of a column by the script could lead to the reproduction of personal data because it was assumed to be a category or column heading, and was therefore included in the output data.
  - All text or category based columns (and any numerical columns with occasional category values that indicate that are missing, for example) should be double-checked to ensure that they only output actual categories that are safe to release (e.g. occupation categories like “teacher”, status categories like “inactive”, missing indicators like “-” or “missing”).
-  **The mere presence of a column or category in a data set may constitute private or sensitive information**, e.g. a synthetic data set reveals or suggests that “this organisation holds this type of data about these types of individuals”, which may not always be appropriate to release.

- **111** If all values of a column are identical (e.g. an 'Age' column for a sample where everyone is 50 years old) then the synthetic data output for that column will be identical with the real data for that column. This could be disclosive.
  - Sometimes more subtle relationships from the original data (e.g. associations of 'Race' and 'Sex' in some kinds of data) *might* be reproduced, and care should be taken to ensure that this does not lead to disclosure.
- **!** In very rare cases it is possible that, by random chance, individual records or significant portions of records present in the original data are seen in the synthetic data, or that the synthetic data contains random correlations that correspond to correlations in the original data.
  - **N.B.** Even if released, neither of these situations would constitute any evidence that the original row or correlation was present in the original data set. The original row or correlation was not used to generate the synthetic row(s), it occurred by coincidence. It may still be alarming to some viewers of the data and could be worth manually removing.

It is likely that this short list doesn't cover all possible ways of accidentally reproducing features of the original data. Because of this we recommend that the disclosure process should be as rigorous and proportionate as possible prior to release of the data.

Misconfiguration of the input data file or the tool may also result in the reproduction of records or significant portions of records.

# Step-by-step instructions

Things you will need to have installed on your computer:

- Python, preferably Python 3.
- Two common libraries (pandas and numpy).
- A software tool for viewing, editing and running Python notebooks, such as VSCode or Jupyter.
- Optionally: the library pyreadstat, if you're intending to read in .sav format files.

The latest version of the tool can be downloaded from [here](#).<sup>3</sup> It will be in the form of a compressed archive file under the header 'Assets'. There are two choices of file, both named Source Code.

- If your operating system is Linux-based, download the file ending in .tar.gz.
- If you are using Windows or Mac OS instead, download the file ending in .zip.
- Using your computer's file manager, go to the directory in which you downloaded the file.
- If your operating system is Linux or Windows, right click on the file and select 'Extract Here' or 'Extract All...' in order to extract the files into an uncompressed directory.
- If you are working in Mac OS, double click on the file and an uncompressed directory containing the files will be created.
- You will find the notebook file BIT+\_ADR\_UK\_synth\_data.ipynb in this directory.

We will now take you through the four steps:

1. Check your system can run it.
2. Point it at your data set.
3. Run it.
4. Check it.



## Check your system can run it

This tool consists of a Python notebook. **It will require a version of Python;** a software tool for viewing, editing and running Python notebooks; and the installation of just two very common Python libraries in order to run.

---

<sup>3</sup> An older version of the tool hosted on Google colab is available from [here](#). If you want to see the notebook quickly then please use the older version. As this is a web-hosted version of the tool, do not attempt to upload any data to the colab space.

## Python version

This notebook assumes that you are using Python 3. Some features may not work as written for Python 2 and so will require customisation. In what follows we'll assume that you already have a version of Python installed in the development environment that you're using.

## Viewing, editing and running Python notebooks

Python notebooks (also known as ipython or Jupyter notebooks) require special software tools so that you can read, edit and run them. There are several options available, such as:

- Jupyter
- VSCode with Jupyter extensions installed

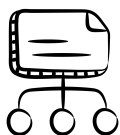
In what follows, we'll assume that you already have one of these tools installed in the development environment you're using.

## Required libraries

This notebook makes use of the following external libraries:

- numpy
- pandas
- pyreadstat (this module is optional and only used if you have Python 3 and wish to process a .sav file)

In some environments, you'll need to install these libraries prior to using the tool. Please refer to the section on Troubleshooting if you need to do this.



## Point it at your data set

Go to *Section 1: Load in data set* in the notebook.

[Figure 1](#) shows the cell where the input file details are set in the tool. You will need to set the value of `file_path`.

- `file_path` should be set to the full path of the file location.

[Figure 2](#) below shows an example of how to specify these variables.

## Important notes

- The supported input data file types are
  - 'csv',
  - 'txt',
  - 'xlsx',
  - 'xls',



- 'sas7bdat',
- 'sav',
- 'dta'
- and 'pkl'.

```
# 1) put in your full file path here, e.g.
"C:\\Users\\FirstName.LastName\\Downloads\\General\\data.csv"

file_path = ""
```

**Figure 1:** Where to specify the input file location.

- The tool will automatically identify the type of file based on its suffix, i.e. a file called example.csv will be identified as a csv file and a file called example.xls will be identified as an Excel file.
- For some formats (such as csv) the tool assumes that the first line of the data file is a header containing the names of each column.
  - **Caution:** If the data file contains no header and the first line is the first record of the data set to be processed, you will need to customise how the tool reads in data files. Otherwise the tool will output the first record as the column headings, which could disclose private data.
  - Similarly, if you wish the tool to start reading in data from a record other than the first (e.g. from the fifth record onwards) the tool will need to be customised.

```
# 1) put in your full file path here, e.g.
"C:\\Users\\FirstName.LastName\\Downloads\\General\\data.csv"

file_path = "..\\data\\california_housing_train.csv"
```

**Figure 2:** Specifying the input file location.

## Advanced Configuration: SQL queries and database pipelines

Sometimes the data you wish to generate synthetic data from will not be available in a file format, because you are reading it in from a database. To access the data you will need to query the database (usually using a language called SQL) and translate the results into

something that Python can understand. This usually requires using an additional library and the functions it contains to communicate with the database.

The precise way this is done will depend on the library you or your organisation prefers to use and the functions that library contains. Because of this, this advice is very general.

- Leave `file_path=""` in the cell identified by [Figure 1](#).
- Uncomment all the code in the two cells under the heading 'Using a data pipeline'.
- Replace LIBRARY with the name of the library you are using.
- Replace PIPELINE\_QUERY with the function that you are using to access the pipeline. This function will usually contain the database query as input.
- Replace OUTPUT\_FILE\_NAME with the name of the file you want to contain the synthetic data. Do not include a file suffix.

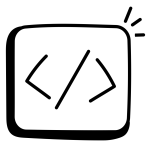
## Advanced Configuration: Linking Datasets

This tool produces synthetic data for single table data sets ('flat files'). If you wish to generate linked data using this tool without modifying it, you can:

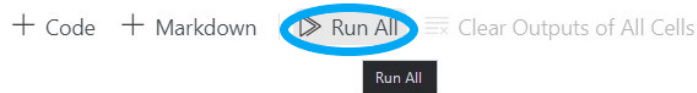
- Link the data sets so that they form a single table, and use this as input for the tool. The output will be a single table that can be decomposed into the component tables.
- Note that this approach will be limited by the size of your computer's memory. If the single table is very large then it may not work.

It is also in principle possible to produce linked synthetic data sets without linking them together first.

- For tables that are linked in a simple fashion where a primary key in one table is a foreign key in another, it should be possible to adapt the tool so that the primary key values from the first table are reproduced with the appropriate frequencies in the foreign key table of the second table. The details of how to do this are beyond the scope of this guide but if you attempt this then please let us know!
- For tables from originally distinct data sources that are linked through a linking table, adapting the tool is more difficult and may have to be tailored to the specifics of the data set at hand. How to do this is also beyond the scope of this guide.



## Run it



# Create a Low-Fidelity S

**Figure 3:** Running all cells in VSCode. Click on the option in the blue circle

The next step is to run all cells in the notebook. There are a number of different ways to do this depending on the environment you're working in (in fact, many environments have several different ways of doing this). Some suggested methods are given below, but if you're very familiar with a particular environment you may have a preferred alternative that you want to use.

- **JupyterLab:** press Esc to go to Command Mode, Ctrl+a to select all cells, Shift+Enter to run all selected cells.
- **VSCode:** click the Run All button, as shown in [Figure 3](#).

Once you know how to run all the cells in the notebook, follow the following steps to generate synthetic data:

- Run all cells in the notebook. For very large data files this may take some time to complete.
- If you are using an Excel spreadsheet as the input file, it is possible that a warning message 'SystemExit: 0' will be issued by the notebook. This indicates that the program is working correctly and has no implications for the results. Go to [this section](#), which discusses how the tool processes Excel spreadsheet data.
- Go to *Section 3: Generate Synthetic Data* in the notebook.
- The data read in from the input file is displayed beneath the cell in [Figure 4](#). Check that it makes sense and matches the data in the original file.
- Information about each column and the data it carries is displayed below the cell that starts with '# save original column names' ([Figure 5](#)). Each column is displayed as an entry in a Python list that begins with the column name and ends with the column type (numeric, categorical, datetime, string, or NA), so that person\_name\_string corresponds to the column 'person\_name' which has the type string.

```
original_data
```

✓ 0.3s

	longitude	latitude	housing
0	-114.31	34.19	
1	-114.47	34.40	
2	-114.56	33.69	
3	-114.57	33.64	

**Figure 4:** Displaying the input data.

```
new_column_names
```

✓ 1.3s

```
['longitude_numeric',
 'latitude_numeric',
 'housing_median_age_categorical',
 'total_rooms_numeric',
 'total_bedrooms_numeric',
 'population_numeric',
 'households_numeric',
 'median_income_numeric',
 'median_house_value_numeric']
```

**Figure 5:** Displaying column classification data.

```

# save to csv
original_data.to_csv(file
# save to sav format if c
if MODE == 'sav_file':
    pyreadstat.write_sav(
        column_labels
        variable_disp

original_data.head()
✓ 0.7s

```

	longitude	latitude	houisi
0	-118.39	32.80	
1	-120.34	34.21	
2	-119.62	35.81	
3	-118.52	36.61	
4	-118.16	37.20	

**Figure 6:** Location of the synthetic data summary.

- If, on examining the output, you disagree with the classification that the algorithm has assigned to the column, you may correct this using the optional step described in the section [Correcting the variable classification](#) below.
- If the tool has executed without any problems, you should see a summary of the first few lines of the synthetic data it has generated below the cell depicted in [Figure 6](#).
- The output file containing the synthetic data will be present as `file_name_synthetic.csv`, where `file_name` is taken from the name of the file in `file_path`, which was set in the section [Setting up the input file](#).

### Advanced Configuration: Saving files in different locations

Sometimes you will need to save the synthetic data to a location other than the current working directory. If you wish to save in a different directory that is on your computer or that your computer can access, modify the variable `data_location` in the final cell as follows:

- `data_location = DIRECTORY_PATH + file_name + '_synthetic'`
- Here, `DIRECTORY_PATH` should be the path to the directory where you want to save the synthetic data. If it's absent (which is the default), the synthetic data will be saved in the same directory as the notebook.

Occasionally, you may need to save to other locations that may not be accessible via a file path (for example, S3 buckets on Amazon Web Services). This might require further

modifications to the notebook that are beyond the scope of this guide: please talk to whoever is in charge of your system for guidance.

## Advanced Configuration: Changing output appearance

The tool gives some limited control over some elements of the synthetic data output. By changing the values of the variables in the cell beneath the heading ‘Modifying the output appearance’ we can

- specify how null or NaN data values appear in the synthetic data output by changing the value of `null_string` (default value: “”).
- change the decimal precision of all real values in the synthetic data output by changing the value of `numerical_precision` (default value: 2).

## Correcting the variable classification

```
# original_data =  
original_data.rename({"column_to_be_renamed1":"new_column_name1","column_to_be_renamed2":"new_column_name2"}, axis='columns')  
# original_data.columns
```

**Figure 7:** Cell containing classification correction code.

[Figure 7](#) shows the cell containing the classification correction code. If you wish to correct the classification of each column:

- First, remove the # symbols at the beginning of each line. This means that the Python interpreter will no longer ignore them.
- Between the braces { and } in the first line, replace the text with “column\_classification”: “column\_new\_classification”, where column is the name of the column, classification is the old classification of the column and new\_classification is the classification you wish to apply to the column.
- An example of how to do this is in [Figure 8](#) below, where we are reassigning the type of the column ‘housing\_median\_age’ from ‘categorical’ to ‘numeric’.
- Rerun the script. If the above has been done correctly, you should see a list of columns beneath this cell containing your corrections.

```
original_data =  
original_data.rename({"housing_median_age_categorical":"housing_median_age_numeric"}, axis='columns')  
original_data.columns
```

**Figure 8:** Example of reassigning the type of a column.

## Generating synthetic data from an Excel spreadsheet

Excel spreadsheets may consist of several different tables on different worksheets. Because of this, unlike other file formats the tool will follow a different path in the notebook. It will output a file named `file_name_synthetic.xls` where `file_name` is taken from the name of the file in `file_path`, which is set in the section [Setting up the input file](#). It will not display any output within the notebook.



### Check it

**Important:** While the synthetic data is not intended to contain any records resembling those in the original data, **there is no automatic guarantee of this**. It is therefore important to scrutinise the output contained in `file_name_synthetic.csv` in order to determine that

- It has not reproduced records that are present in the original data set by chance.
- It has not accidentally reproduced any correlations present in the original data set.
- Sensitive or identifying information present in the dataset has not been included due to misconfiguration of the tool or for other reasons.
- If there is a column where all values are identical, this column will be identical in both the synthetic and real data. If it is the case that this contains information about individuals (e.g. ages in a dataset where all records have a value of 50), this is potentially disclosive.
- It's possible that the tool may have misidentified some columns. For example, if there are a large number of possible values in a categorical field, the tool may misinterpret the field as being a string or numerical field. To fix this issue, follow the instructions in the section [Correcting the Variable Classification](#).

The Background section of this guide discusses [why you should have a disclosure control process in place to ensure that the synthetic data does not accidentally reveal sensitive information](#). This process should be followed to ensure that the synthetic data can be released safely.

# Troubleshooting

## Installing missing libraries

This may vary depending on the environment you're using. There are two package managers that allow you to install Python libraries, pip and conda. Which to use depends on how Python has been installed. If it was installed using conda, then you should install packages using conda and not pip.

If you don't know which was used to install Python, check with whoever is responsible for managing your system. They may also be able to install missing libraries for you!

### Using pip

To check if numpy and pandas are installed, open a command line terminal window and type:

```
python -m pip freeze
```

This will list all the installed Python libraries. If numpy and pandas are on this list, you don't have to do anything else. If either or both aren't on the list, do the following:

For any libraries that are not installed, you should run from the command line:

```
python -m pip install MISSING_LIBRARY_NAME
```

from the command line in order to install them. We recommend setting up and using a [virtual environment](#) before installing libraries wherever possible in order to make sure that you're always using the same versions of the same libraries for a given task.

### Using conda

To check if numpy and pandas are installed, open a command line terminal window and type:

```
conda list
```

If numpy and pandas appear on this list, they're installed and you don't have to do anything else. If either or both are not on the list, then for each missing library run

```
conda install MISSING_LIBRARY_NAME
```

This should install the missing libraries.

## Library compatibility

Older versions of the pandas library are incompatible with more recent versions of the numpy library. If this problem arises, there are two possible solutions:



- *If you are not using a virtual environment:* Uncomment the Python code in the cell under the heading 'Automated package installation (optional)'. Running the notebook should now check the compatibility of your libraries before downloading and installing compatible versions.
- *If you are using a virtual environment:* Using the requirements.txt file included in the tool repository, type

```
python -m pip install -r requirements.txt
```

Note that you may have to uninstall the previously installed versions of the libraries first.

## For advanced users

### Can I use a Python script instead of a notebook?

For some workflows using a Python script may be more convenient than using a notebook. JupyterLab and VSCode all provide options for converting an iPython notebook such as this tool to a Python script.

### Description of functions

In this section we provide a brief overview of the functions defined in Section 2 of the notebook. This is intended to be useful for users seeking a deeper understanding of the tool or who wish to customise it.

`check_if_datetime(x)`

#### Input

x: Pandas Series

#### Return

Boolean: True if x is datetime type, False otherwise.

`check_if_numeric(x)`

#### Input

x: Pandas Series

#### Return

Boolean: True if x is numerical type, False otherwise.

`identify_variable_type(x)`

#### Input

x: Pandas series

#### Return

string: "NA" if x is empty; "categorical" if x contains categorical data, "numerical" if x contains float or integer data, "datetime" if x contains datetime date, "string" otherwise.

`prepend(list, str)`

**Input**

list: List of strings

str: string

**Return**

List of strings: each string in list prepended with str.

`generate_datetime(min_time, max_time)`

**Input**

min\_time: datetime

max\_time: datetime

**Return**

datetime: randomly chosen datetime between min\_time (earliest) and max\_time (latest). Currently uses a uniform distribution.

`paste0(string, values)`

**Input**

string: string

values: List of numerical values

**Return**

List of strings: each value in values appended to string (equivalent to R's paste0 function).

`create_synthetic_data(x)`

**Input**

x: Pandas Series

**Return**

Pandas Series: contains the randomly generated contents of the synthetic data column corresponding to x depending on the variable type of x.

- NA: An empty series.
- Numeric: Data generated from a normal distribution obtained by extracting the mean and standard deviation of the original data.

- Categorical: Generated from probability distributions obtained by a cross-tabulation of categories and their observed frequencies.
- Datetime: Date generated from a uniform distribution between the earliest and latest data recorded.
- String:
  - If entries have roughly the same length, we assume that they have a shared pattern (e.g. they are a postcode or IP address). For each character position in the strings we calculate the distribution of characters and from that randomly select a character for that position in a given entry.
  - If not, we replace the string with placeholder text generated from the phrase 'sample text'.