lab1

March 18, 2024

1 LAB 01 - Python version

1.0.1 Disclaimer

The purpose of creating this material is to enhance the knowledge of students who are interested in learning how to solve problems presented in laboratory classes using Python. This decision stems from the observation that some students have opted to utilize Python for tackling exam projects in recent years.

1.0.2 General information

To solve these exercises using Python, you need to install Python (version 3.9.6 or later) and some libraries using pip or conda.

Here's a list of the libraries needed for this case:

- `os`: Provides operating system dependent functionality, commonly used for file operations such as reading and writing files, interacting with the filesystem, etc.
- `pandas`: A data manipulation and analysis library that offers data structures and functions to efficiently work with structured data.
- `numpy`: A numerical computing library that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- `matplotlib.pyplot`: A plotting library for creating visualizations like charts, graphs, histograms, etc.
- `nltk`: The Natural Language Toolkit, a library for natural language processing tasks such as tokenization, stemming, part-of-speech tagging, and more.
- `TfidfVectorizer` from `sklearn.feature_extraction.text`: A utility to convert a collection of text documents into a matrix of TF-IDF features, commonly used for text analysis and machine learning tasks.
- `xlrd`: A Python library used for reading data and formatting information from Excel files (.xls and .xlsx formats). It provides functionality to extract data from Excel worksheets, including cells, rows, columns, and formatting details.

You can download Python from here and follow the installation instructions for your operating system.

For installing libraries using pip or conda, you can use the following commands:

- Using pip:

```
pip install pandas numpy matplotlib nltk scikit-learn xlrd
```

- Using conda:

  ```
  conda install pandas numpy matplotlib nltk scikit-learn xlrd
  ```

Make sure to run these commands in your terminal or command prompt after installing Python. You can also execute them in a cell of a Jupyter Notebook file (`.ipynb`) by starting the command with '!'.

2 Exercise 1

Import some libraries

[1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

2.1 Read file excel

To read the Excel file using a function integrated into the pandas library, you can use the `pd.read_excel()` function. Rewrite the instruction with the argument as the path of the file to be read

[2]:
```
dataset = pd.read_excel("/Users/luca/Library/Mobile Documents/
↪com~apple~CloudDocs/Business Intelligence per Big Data/Laboratories/LAB01/
↪Lab1Materiale/UsersSmall.xls")
```

In a Jupyter Notebook cell, you can print a subset of the representation by simply calling the name of the variable containing the DataFrame.

[3]:
```
dataset
```

[3]:
```
      Age         Workclass  Education      Marital Status         Occupation  \
0     -15         State-gov  Bachelors       Never-married       Adm-clerical
1     150           Private        PhD       Never-married    Exec-managerial
2      39         State-gov  Bachelors       Never-married       Adm-clerical
3      50  Self-emp-not-inc  Bachelors  Married-civ-spouse    Exec-managerial
4      38           Private    HS-grad            Divorced  Handlers-cleaners
..    …               …        …                 …               …
297    65           Private    HS-grad  Married-civ-spouse   Transport-moving
298    37      Self-emp-inc  Bachelors            Divorced              Sales
299    39                 ?    Masters  Married-civ-spouse                  ?
300    24           Private    HS-grad       Never-married       Craft-repair
301    38           Private    HS-grad            Divorced              Sales

       Relationship                Race     Sex Native Country  Response
0     Not-in-family               White    Male  United-States  Negative
1                 ?               White  Female        Jamaica  Negative
2     Not-in-family               White    Male  United-States  Negative
```

```
3          Husband                White    Male  United-States  Negative
4    Not-in-family                White    Male  United-States  Negative
..             …                    …     …            …        …
297        Husband                White    Male  United-States  Negative
298  Not-in-family                White  Female  United-States  Negative
299           Wife  Asian-Pac-Islander  Female              ?  Negative
300      Own-child                White    Male  United-States  Negative
301  Not-in-family                White    Male  United-States  Negative

[302 rows x 10 columns]
```

2.2 How to handle Missing values?

Find if there are missing values.

Usually in a real dataset the missing values are stored with a nan value. In this case we have ? as missing values representation.

So first of all we can replace each '?' symbol in a nan value. The we will apply some important and classical functions.

[4]:
```
dataset.replace(to_replace = '?', value = np.nan, inplace = True)
```

[5]:
```
dataset.isnull().sum() # count the number of missing values for each column
```

[5]:
```
Age                0
Workclass         16
Education          0
Marital Status     0
Occupation        16
Relationship       1
Race               0
Sex                0
Native Country     8
Response           0
dtype: int64
```

As you have seen in class there are different methodologies for filling the nan values. Here we will use the average for the numerical data and the most frequent string for non-numerical columns

[6]:
```
# Replace NaN values with the average value for numerical columns
for col in dataset.select_dtypes(include=np.number).columns:
    dataset[col].fillna(dataset[col].mean(), inplace=True) # Get the average␣
 ↪value for the column and replace NaN values with it

# Replace NaN values with the most frequent value for non-numerical columns
for col in dataset.select_dtypes(exclude=np.number).columns:
    mode_val = dataset[col].mode()[0]  # Get the most frequent string value
```

```
    dataset[col].fillna(mode_val, inplace=True) # Get the most frequent value␣
↪for the column and replace NaN values with it
```

[7]:
```
dataset
```

[7]:
```
     Age         Workclass  Education      Marital Status         Occupation  \
0    -15         State-gov  Bachelors       Never-married       Adm-clerical
1    150           Private        PhD       Never-married    Exec-managerial
2     39         State-gov  Bachelors       Never-married       Adm-clerical
3     50  Self-emp-not-inc  Bachelors  Married-civ-spouse    Exec-managerial
4     38           Private    HS-grad            Divorced  Handlers-cleaners
..   …               …        …                 …               …
297   65           Private    HS-grad  Married-civ-spouse   Transport-moving
298   37      Self-emp-inc  Bachelors            Divorced              Sales
299   39           Private    Masters  Married-civ-spouse     Prof-specialty
300   24           Private    HS-grad       Never-married       Craft-repair
301   38           Private    HS-grad            Divorced              Sales

      Relationship                Race     Sex Native Country  Response
0    Not-in-family               White    Male  United-States  Negative
1          Husband               White  Female        Jamaica  Negative
2    Not-in-family               White    Male  United-States  Negative
3          Husband               White    Male  United-States  Negative
4    Not-in-family               White    Male  United-States  Negative
..             …                   …     …            …       …
297        Husband               White    Male  United-States  Negative
298  Not-in-family               White  Female  United-States  Negative
299           Wife  Asian-Pac-Islander  Female  United-States  Negative
300      Own-child               White    Male  United-States  Negative
301  Not-in-family               White    Male  United-States  Negative

[302 rows x 10 columns]
```

We can now check the number of missing values in the dataset.

We can see that there are no missing values in the dataset.

[8]:
```
dataset.isnull().sum()
```

[8]:
```
Age               0
Workclass         0
Education         0
Marital Status    0
Occupation        0
Relationship      0
Race              0
Sex               0
Native Country    0
```

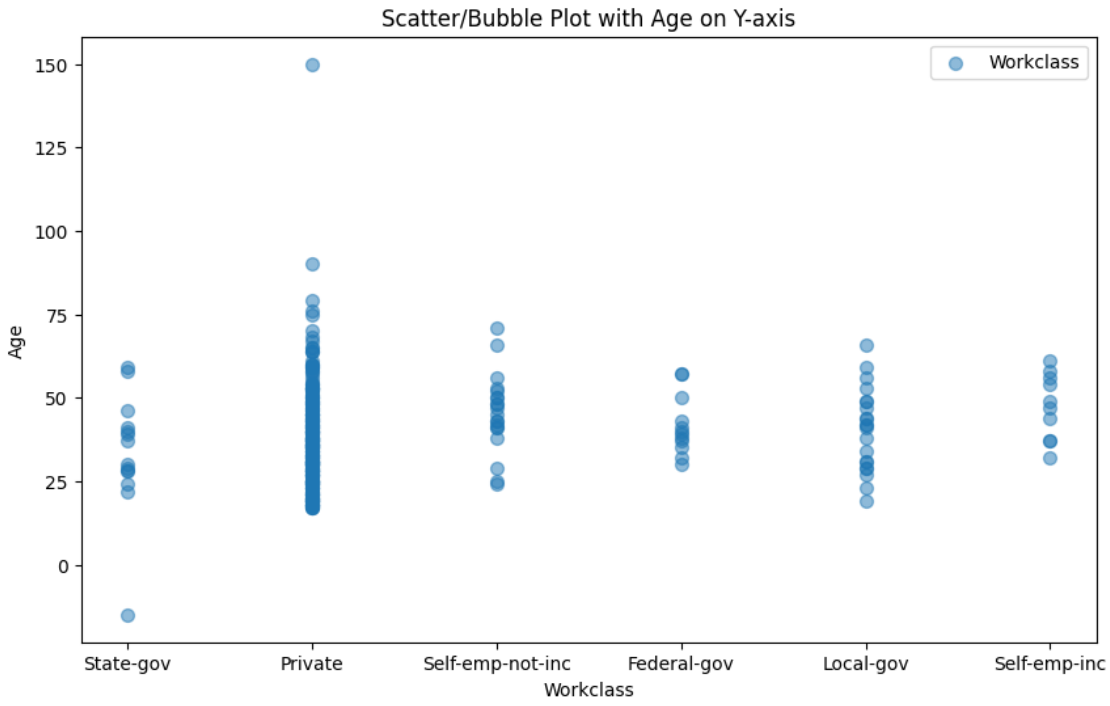
```
Response          0
dtype: int64
```

2.3 Outlier detection

You can plot a scatter/bubble plot to identify some outliers

[9]:
```
# Fix the 'Age' attribute on the y-axis
age_values = dataset['Age']

# Plot scatter/bubble plot with an attribute on the x-axis. Ypu caan choose␣
 ↪what ever attribute you want
attribute = "Workclass"
plt.figure(figsize=(10, 6))
plt.scatter(dataset[attribute], age_values, s=50, alpha=0.5, label=attribute)
plt.xlabel(attribute, fontsize=10)  # Adjusted x-axis label font size
plt.ylabel('Age')
plt.title('Scatter/Bubble Plot with Age on Y-axis')
plt.legend()
plt.show()
```



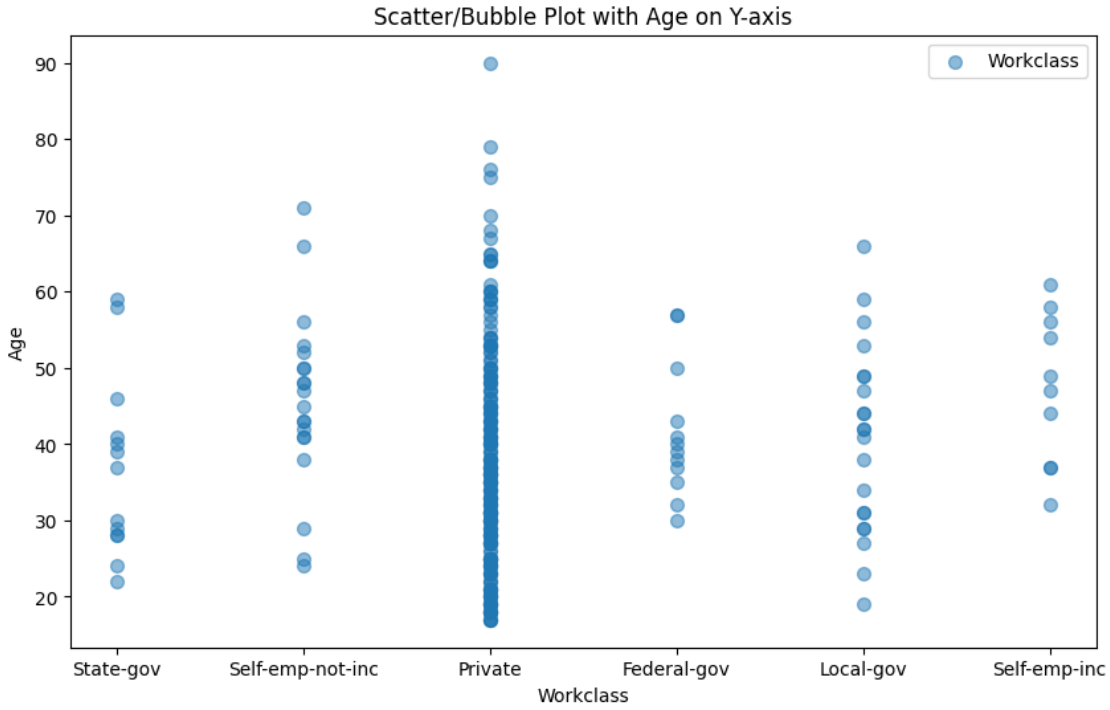
As evident, the 'Age' attribute in our dataset contains errors, such as improbable values like 150 for age. To ensure the integrity of our data, we need to perform cleaning by filtering out such rows from the dataset.

[10]:
```
condition = (dataset['Age'] >= 0) & (dataset['Age'] < 105) # Get the condition␣
↪for the age values (between 0 and 105 years old)
dataset = dataset[condition] # Apply the condition to the dataset and store the␣
↪result in the dataset variable
```

We can plot again the same graph created in the previous step and examine it to identify any differences.

[11]:
```
# Fix the 'Age' attribute on the y-axis
age_values = dataset['Age']

# Plot scatter/bubble plot with an attribute on the x-axis. Ypu caan choose␣
↪what ever attribute you want
attribute = "Workclass"
plt.figure(figsize=(10, 6))
plt.scatter(dataset[attribute], age_values, s=50, alpha=0.5, label=attribute)
plt.xlabel(attribute, fontsize=10)
plt.ylabel('Age')
plt.title('Scatter/Bubble Plot with Age on Y-axis')
plt.legend()
plt.show()
```



2.4 Discretize some data

Data discretization is a preprocessing technique used to transform continuous data into discrete intervals or categories. This process involves dividing the continuous range of values into a finite number of intervals, or bins. Discretization is commonly used in data analysis and machine learning tasks to simplify data representation, reduce noise, and improve the performance of algorithms.

[12]:
```
# Define bin edges
bin_edges = [0, 18, 30, 40, 50, float('inf')]  # Define your own bin edges as␣
 ↪needed
# Define bin labels
bin_labels = ['0-18', '19-30', '31-40', '41-50', '51+']  # Define labels for␣
 ↪each bin
# Discretize 'Age' attribute using cut() function
dataset['Age'] = pd.cut(dataset['Age'], bins=bin_edges, labels=bin_labels,␣
 ↪right=False)
```

```
/var/folders/rb/rpwh82c933b_w9vzlcm_hzpc0000gn/T/ipykernel_89048/4213913650.py:6
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataset['Age'] = pd.cut(dataset['Age'], bins=bin_edges, labels=bin_labels,
right=False)
```

[13]:
```
dataset
```

[13]:

	Age	Workclass	Education	Marital Status	\
2	31-40	State-gov	Bachelors	Never-married	
3	51+	Self-emp-not-inc	Bachelors	Married-civ-spouse	
4	31-40	Private	HS-grad	Divorced	
5	51+	Private	11th	Married-civ-spouse	
6	19-30	Private	Bachelors	Married-civ-spouse	
..	…	…	…	…	
297	51+	Private	HS-grad	Married-civ-spouse	
298	31-40	Self-emp-inc	Bachelors	Divorced	
299	31-40	Private	Masters	Married-civ-spouse	
300	19-30	Private	HS-grad	Never-married	
301	31-40	Private	HS-grad	Divorced	

	Occupation	Relationship	Race	Sex	\
2	Adm-clerical	Not-in-family	White	Male	
3	Exec-managerial	Husband	White	Male	
4	Handlers-cleaners	Not-in-family	White	Male	
5	Handlers-cleaners	Husband	Black	Male	
6	Prof-specialty	Wife	Black	Female	

```
..                  …            …                   …     …
297   Transport-moving        Husband                White    Male
298              Sales  Not-in-family                White  Female
299     Prof-specialty           Wife  Asian-Pac-Islander  Female
300       Craft-repair      Own-child                White    Male
301              Sales  Not-in-family                White    Male

    Native Country  Response
2    United-States  Negative
3    United-States  Negative
4    United-States  Negative
5    United-States  Negative
6             Cuba  Negative
..             …       …
297  United-States  Negative
298  United-States  Negative
299  United-States  Negative
300  United-States  Negative
301  United-States  Negative

[300 rows x 10 columns]
```

3 Exercise 2

Please be aware that the following code utilizes several libraries and may appear complex. If you encounter any difficulties in understanding the code, feel free to reach out for clarification or assistance

[14]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
```

3.0.1 This code could be divided in different sections:

Download NLTK resources

- The code begins by downloading necessary resources from NLTK (Natural Language Toolkit) library, specifically the 'punkt' tokenizer and 'stopwords' corpus.

Define Folder Path and Initialize Variables

- Next, the code defines the folder path containing the text files to be processed.
- It initializes two lists: 'preprocessed_texts' to hold preprocessed text from each file, and 'file_names' to store the names of the files.

Initialize Snowball Stemmer and Define Italian Stopwords

- Italian stopwords are defined to remove common and uninformative words from the text. Stopwords are either loaded from NLTK's stopwords corpus or from a custom file.

Loop Through Files in the Folder

- The code iterates through each file in the specified folder.
- It reads the content of each file, tokenizes the text into individual words, converts them to lowercase, removes stopwords, and performs stemming on the remaining tokens.
- The preprocessed text is then joined back together and appended to the 'preprocessed_texts' list. Additionally, the file name is added to the 'file_names' list.

TF-IDF Vectorization

- The TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer is initialized to convert the preprocessed text data into a matrix of TF-IDF features.
- The preprocessed text data is fitted and transformed using the TF-IDF vectorizer, resulting in a TF-IDF matrix.
- The TF-IDF matrix is converted into a DataFrame for better visualization, with columns representing unique features extracted from the text and rows corresponding to the files processed.

[15]:
```
# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

# Define the folder path containing the text files
folder_path = "/Users/luca/Library/Mobile Documents/com~apple~CloudDocs/
 ↪Business Intelligence per Big Data/Laboratories/LAB01/Lab1Materiale/
 ↪wikipedia"
# List to hold preprocessed text from each file
preprocessed_texts = []
# List to hold file names
file_names = []
# Initialize Snowball stemmer for Italian
stemmer = SnowballStemmer("italian")
# Define Italian stopwords
# if you want to use the library: italian_stopwords = set(stopwords.
 ↪words('italian'))
# else you can use a file with the stopwords
file_italian_stopwords = open("/Users/luca/Library/Mobile Documents/
 ↪com~apple~CloudDocs/Business Intelligence per Big Data/Laboratories/LAB01/
 ↪Lab1Materiale/stopwordsEnglish.txt", "r")
italian_stopwords = set(file_italian_stopwords.read().splitlines())

# Loop through files in the folder
for file_name in os.listdir(folder_path):
    file_path = os.path.join(folder_path, file_name)
```

```
    with open(file_path, 'r', encoding='utf-8') as file:
        # Read the text from the file
        text = file.read()
        # Tokenization
        tokens = word_tokenize(text)
        # Transform to lowercase
        tokens_lower = [token.lower() for token in tokens]
        # Remove stopwords
        tokens_filtered = [token for token in tokens_lower if token not in␣
 ↪italian_stopwords]
        # Stemming
        tokens_stemmed = [stemmer.stem(token) for token in tokens_filtered]
        # Join the tokens back to form preprocessed text
        preprocessed_text = ' '.join(tokens_stemmed)
        # Append the preprocessed text to the list
        preprocessed_texts.append(preprocessed_text)
        # Append the file name to the list
        file_names.append(file_name)

# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
# Fit and transform the preprocessed text data
tfidf_matrix = tfidf_vectorizer.fit_transform(preprocessed_texts)

# Convert the TF-IDF matrix to a DataFrame for better visualization
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.
 ↪get_feature_names_out(), index=file_names)
tfidf_df
```

```
[nltk_data] Downloading package punkt to /Users/luca/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/luca/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[15]:

	10	1154	1300	1564	1642	16th	\
text9.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text8.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text6.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text7.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text5.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text4.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text12.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text1.txt	0.050967	0.000000	0.000000	0.050967	0.050967	0.000000	
text11.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text3.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.077324	
text2.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text10.txt	0.000000	0.037299	0.037299	0.000000	0.000000	0.000000	

```
               1777      17th      1809     1820s  …      word      work  \
text9.txt   0.000000  0.000000  0.000000  0.000000  …  0.000000  0.000000
text8.txt   0.000000  0.000000  0.000000  0.000000  …  0.000000  0.000000
text6.txt   0.000000  0.000000  0.000000  0.000000  …  0.000000  0.000000
text7.txt   0.000000  0.000000  0.000000  0.046985  …  0.087032  0.000000
text5.txt   0.000000  0.000000  0.000000  0.000000  …  0.039510  0.000000
text4.txt   0.000000  0.033419  0.000000  0.000000  …  0.000000  0.000000
text12.txt  0.000000  0.000000  0.000000  0.000000  …  0.000000  0.000000
text1.txt   0.050967  0.000000  0.050967  0.000000  …  0.031470  0.043771
text11.txt  0.000000  0.000000  0.000000  0.000000  …  0.000000  0.000000
text3.txt   0.000000  0.033203  0.000000  0.000000  …  0.095487  0.033203
text2.txt   0.000000  0.000000  0.000000  0.000000  …  0.018258  0.000000
text10.txt  0.000000  0.000000  0.000000  0.000000  …  0.000000  0.000000

               works     world   written       yal      yams     years  \
text9.txt   0.000000  0.179252  0.000000  0.000000  0.058062  0.118610
text8.txt   0.000000  0.024350  0.000000  0.000000  0.000000  0.000000
text6.txt   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
text7.txt   0.000000  0.029011  0.000000  0.093969  0.000000  0.000000
text5.txt   0.000000  0.039510  0.000000  0.000000  0.000000  0.000000
text4.txt   0.038913  0.000000  0.033419  0.000000  0.000000  0.026497
text12.txt  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
text1.txt   0.000000  0.000000  0.131314  0.000000  0.000000  0.034706
text11.txt  0.000000  0.044091  0.000000  0.000000  0.000000  0.024312
text3.txt   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
text2.txt   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
text10.txt  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

              yield   zealand
text9.txt   0.000000  0.000000
text8.txt   0.000000  0.000000
text6.txt   0.000000  0.000000
text7.txt   0.000000  0.046985
text5.txt   0.000000  0.000000
text4.txt   0.000000  0.000000
text12.txt  0.000000  0.000000
text1.txt   0.000000  0.000000
text11.txt  0.000000  0.000000
text3.txt   0.000000  0.000000
text2.txt   0.000000  0.000000
text10.txt  0.037299  0.000000

[12 rows x 1351 columns]
```