Національний авіаційний університет

Факультет комп’ютерних наук та технологій

**Лабораторна робота №1**

Дисципліна: «Високорівневі мови програмування»

Кафедра: прикладної математики

ОС: бакалавр

Спеціальність: 113 «Прикладна математика»

ОПП: «Прикладне програмне забезпечення»

Виконав: здобувач вищої освіти 3 курсу. 351 групи

Архіпов О.Т.

Перевірив: Сорокопуд В.І.

Київ 2023

**Зміст**

1. Завдання
2. Теоретична частина
3. Результат роботи програми
4. Висновок
5. Додаток

**Завдання**

1. Створити графічний застосунок для роботи з даними з використанням технологій PyQt5, Numpy, Pandas та MatPlotLib.
2. Розробити макет графічного застосунку в редакторі та зберегти його в ui файл. Згенерувати py файл з ui файлу.
3. Застосунок повинен мати функціонал для завантаження файлів з даними. Відображати їх у вигляді таблиці, малювати графіки по завантаженим даним та показувати основі статистичні показники по цім даним.

**Теоретична частина**

Рandas — програмна бібліотека, написана для мови програмування Python для маніпулювання даними та їхнього аналізу. Вона, зокрема, пропонує структури даних та операції для маніпулювання чисельними таблицями та часовими рядами.

Можливості бібліотеки:

* Об'єкт DataFrame із вбудованим індексуванням для маніпулювання даними.
* Інструменти для зчитування та записування даних між структурами даних у пам'яті та різними форматами файлів.
* Вирівнювання даних та вбудована підтримка пропущених даних.
* Переформатовування для отримання зведених наборів даних.
* Отримання зрізів за мітками, індексування з розширеними можливостями та отримання піднаборів з великих наборів даних.
* Вставляння та вилучення стовпчиків у структурах даних.
* Рушій групування, що дозволяє робити з наборами даних операції розділення-зміни-об'єднання.
* Злиття та з'єднання наборів даних.
* Ієрархічне індексування осей для роботи з даними високої вимірності в структурі даних нижчої вимірності.
* Функціональність для часових рядів: породження діапазонів дат та перетворення частоти, статистики рухливого вікна, лінійні регресії рухливого вікна, зсування дат та запізнювання.

Pandas в основному використовується для аналізу даних. Pandas дозволяє імпортувати дані з різних форматів файлів, таких як значення, розділені комами, JSON, SQL, Microsoft Excel. Pandas дозволяє здійснювати різні операції з обробкою даних, такі як об'єднання, зміна форми, вибір, а також очищення даних та функції перегляду даних.

**Numpy** — розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами.

Оскільки Python — інтерпретована мова, математичні алгоритми, часто працюють в ньому набагато повільніше ніж у компільованих мовах, таких як C або навіть Java. NumPy намагається вирішити цю проблему для великої кількості обчислювальних алгоритмів забезпечуючи підтримку багатовимірних масивів і безліч функцій і операторів для роботи з ними. Таким чином будь-який алгоритм, який може бути виражений в основному як послідовність операцій над масивами і матрицями, працює так само швидко, як еквівалентний код, написаний на C.

NumPy можна розглядати як гарну вільну альтернативу MATLAB, оскільки мова програмування MATLAB зовні нагадує NumPy: обидві вони інтерпретовані, і обидві дозволяють користувачам писати швидкі програми поки більшість операцій проводяться над масивами або матрицями, а не над скалярами. Перевага MATLAB у великій кількості доступних додаткових тулбоксів, включаючи такі як пакет Simulink. Основні пакети, що доповнюють NumPy, це: SciPy — бібліотека, що додає більше MATLAB-подібної функціональності; Matplotlib — пакет для створення графіки в стилі MATLAB.

**matplotlib** — бібліотека на мові програмування Python для візуалізації даних двовимірною 2D графікою (3D графіка також підтримується). Отримувані зображення можуть бути використані як ілюстрації в публікаціях.

Matplotlib є гнучким, легко конфігурованим пакетом, який разом з NumPy, SciPy і IPython надає можливості, подібні до MATLAB. В даний час пакет працює з декількома графічними бібліотеками, включаючи wxWindows і PyGTK.

Пакет підтримує багато видів графіків і діаграм:

* Графіки (line plot)
* Діаграми розсіювання (scatter plot)
* Стовпчасті діаграми (bar chart) і гістограми (histogram)
* Секторні діаграми (pie chart)
* Діаграми «Стовбур-листя» (stem plot)
* Контурні графіки (contour plot)
* Поля градієнтів (quiver)
* Спектральні діаграми (spectrogram)

Користувач може вказати осі координат, сітку, додати підписи і пояснення, використовувати логарифмічну шкалу або полярні координати.

Нескладні тривимірні графіки можна будувати з допомогою набору інструментів (toolkit) **mplot3d**. Існують і інші набори інструментів: для картографії, для роботи з Excel, утиліти для GTK та інші.

З допомогою Matplotlib можна створювати і анімовані зображення.

**Результат роботи програми**

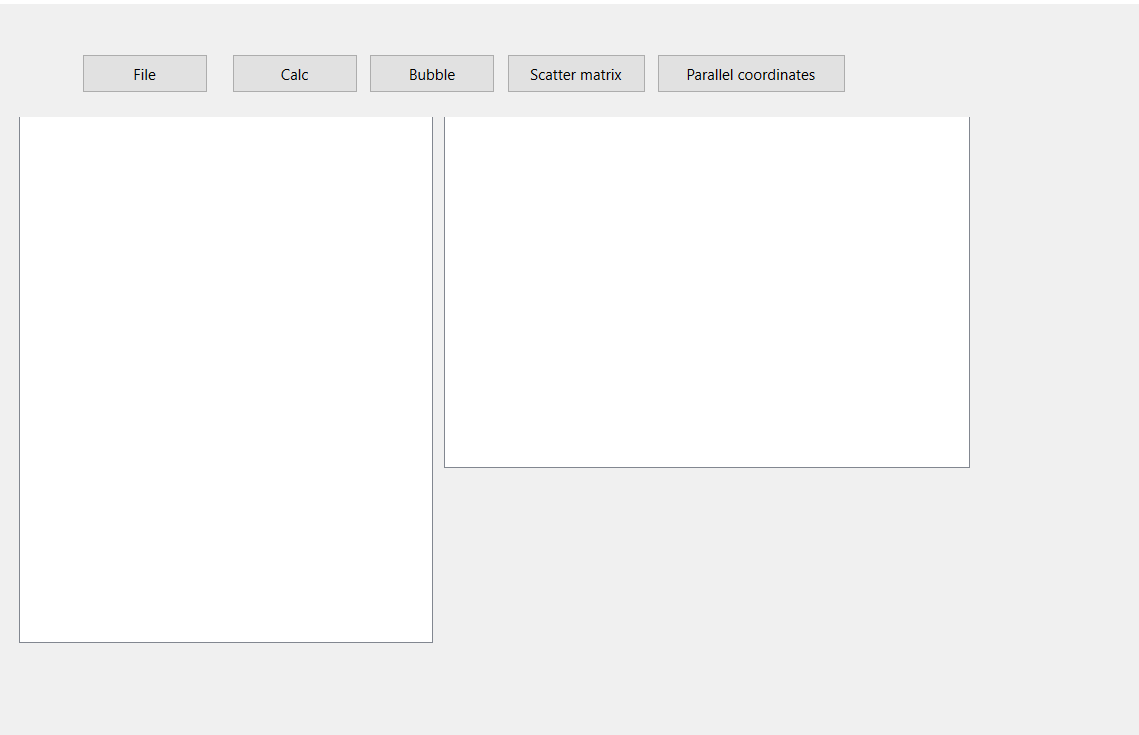


Рисунок 1. Вікно програми

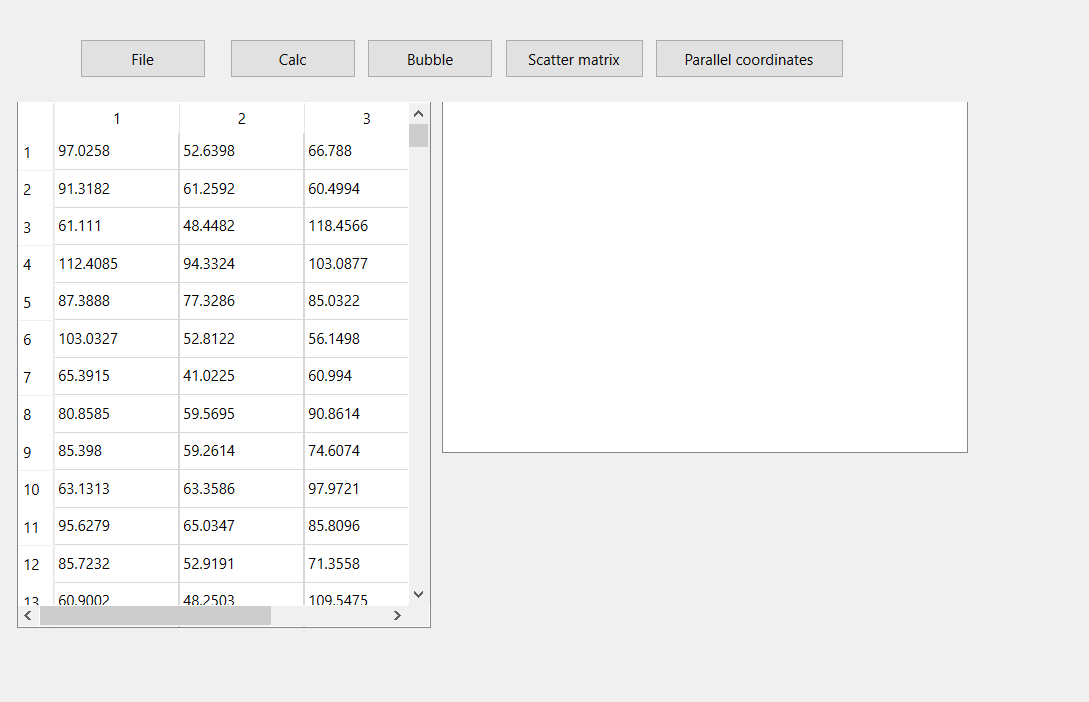


Рисунок 2. Зчитані дані

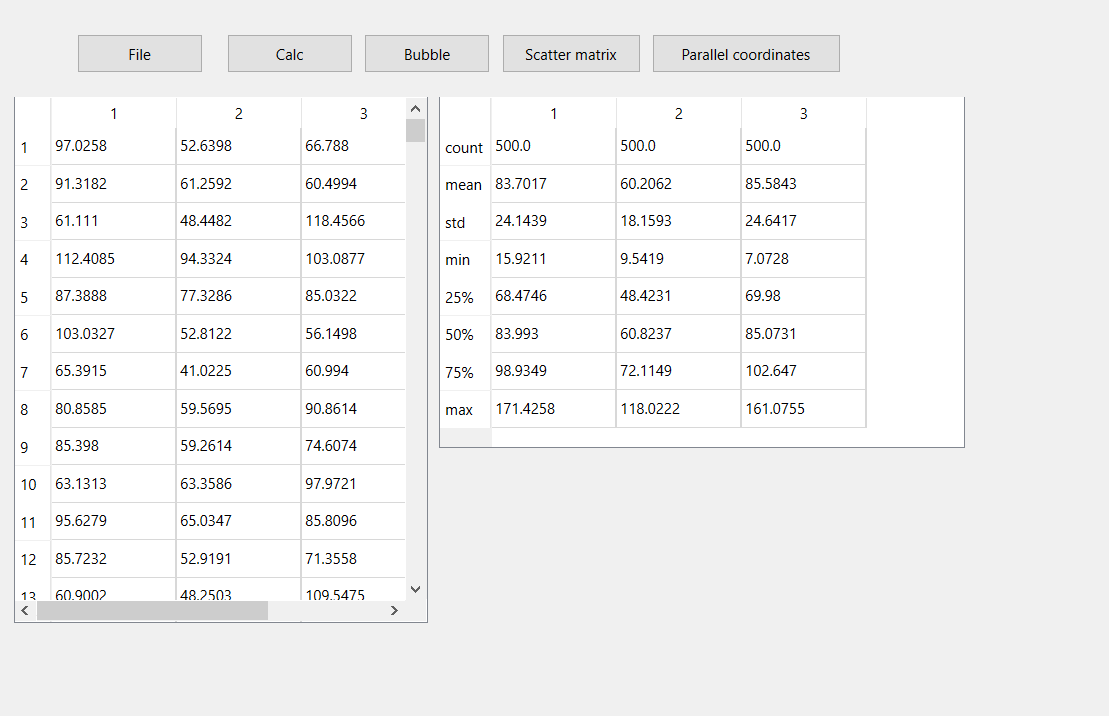


Рисунок 3. Обрахунок статистик

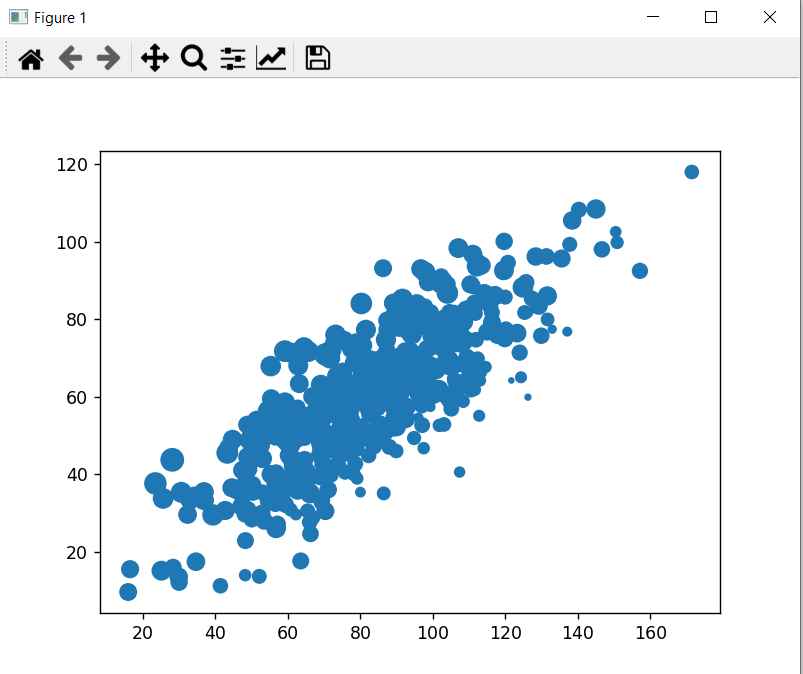


Рисунок 4. Бульбашкова діаграма

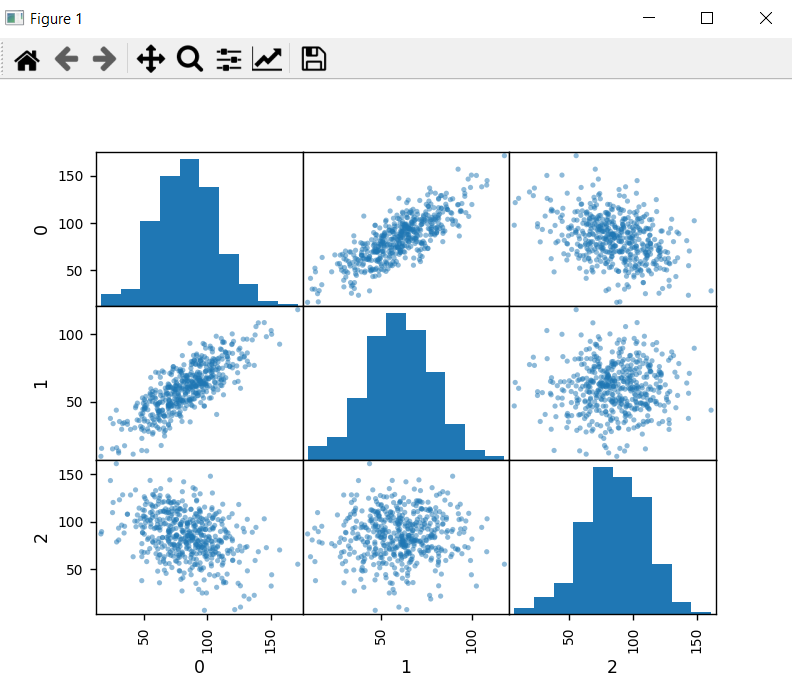


Рисунок 5. Матриця діаграм розкиду

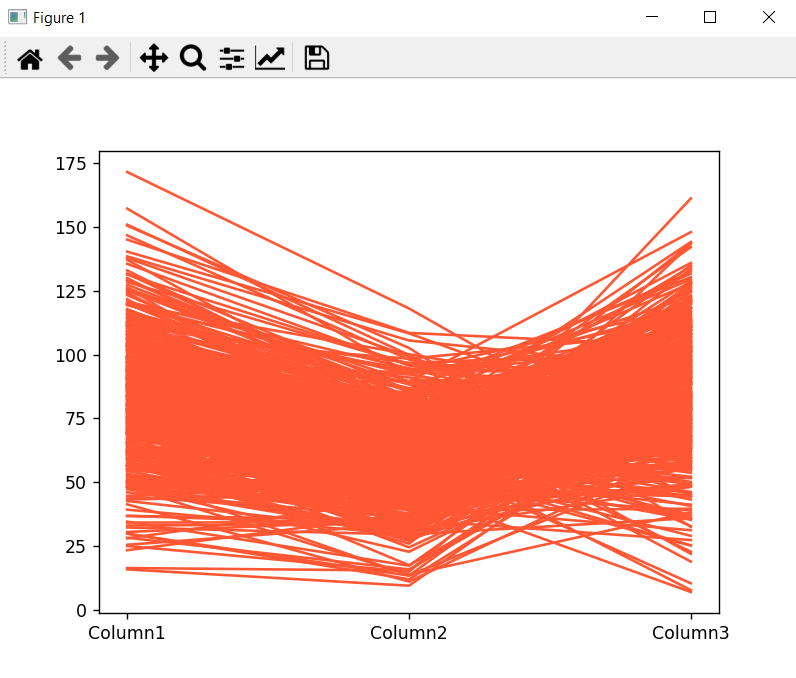


Рисунок 6. Графік паралельних координат

**Висновок**

1. Створив графічний застосунок для роботи з даними з використанням технологій PyQt5, Numpy, Pandas та MatPlotLib.
2. Розробив макет графічного застосунку в редакторі та зберіг його в ui файл. Згенерував py файл з ui файлу.
3. Застосунок має функціонал для завантаження файлів з даними. Відображає їх у вигляді таблиці, малює графіки по завантаженим даним та показує основі статистичні показники по цім даним.

**Додаток**

App.py

import sys

from PySide6.QtWidgets import QApplication, QWidget

from main import Window

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = Window()

    window.show()

    sys.exit(app.exec())

main.py

from PySide6.QtWidgets import QWidget, QFileDialog

from PySide6.QtGui import QStandardItemModel, QStandardItem

from ui\_form import Ui\_Widget

import pandas as pd

import matplotlib.pyplot as plt

class Window(QWidget):

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        self.ui = Ui\_Widget()

        self.ui.setupUi(self)

        self.ui.btn\_file.clicked.connect(self.Read)

        self.ui.btn\_calc.clicked.connect(self.Calc\_statistics)

        self.ui.btn\_bubble.clicked.connect(self.Draw\_bubble)

        self.ui.btn\_scatter\_matrix.clicked.connect(self.Scatter\_plot)

        self.ui.btn\_par\_coord.clicked.connect(self.Par\_coord)

        self.columns = [[]]

    def Read(self):

        file, checked = QFileDialog.getOpenFileName(self, "Open File", "", "Text Files (\*.txt)")

        if file:

            with open(file, 'r') as file1:

                lines = file1.readlines()

            columns = [[] for \_ in range(len(lines[0].split()))]

            for line in lines:

                line = line.strip()

                values = line.split()

                for i, value in enumerate(values):

                 columns[i].append(float(value))

            self.columns = columns

        table\_view = self.ui.tableView

        if table\_view.model() is not None:

            table\_view.setModel(None)

        model = QStandardItemModel()

        num\_rows = len(self.columns[0])

        num\_cols = len(self.columns)

        model.setRowCount(num\_rows)

        model.setColumnCount(num\_cols)

        for i, row in enumerate(self.columns):

            for j, item in enumerate(row):

                model.setData(model.index(j, i), str(item))

        self.columns = list(zip(\*self.columns))

        table\_view.setModel(model)

        table\_view.show()

    def Calc\_statistics(self):

        df = pd.DataFrame(self.columns)

        description = df.describe()

        table\_view = self.ui.tableView\_2

        if table\_view.model() is not None:

            table\_view.setModel(None)

        model = QStandardItemModel()

        model.setColumnCount(len(description.columns))

        model.setRowCount(len(description.index))

        for i, (index, row) in enumerate(description.iterrows()):

            model.setVerticalHeaderItem(i, QStandardItem(index))

            for j, value in enumerate(row):

                item = QStandardItem(str(round(value,4)))

                model.setItem(i, j, item)

        table\_view.setModel(model)

        table\_view.show()

    def Draw\_bubble(self):

        self.columns = list(zip(\*self.columns))

        if len(self.columns) == 3:

            plt.scatter(self.columns[0], self.columns[1], s=self.columns[2])

            plt.show()

        self.columns = list(zip(\*self.columns))

    def Scatter\_plot(self):

        df = pd.DataFrame(self.columns)

        pd.plotting.scatter\_matrix(df)

        plt.show()

    def Par\_coord(self):

        fig, ax = plt.subplots()

        for i, col in enumerate(self.columns):

            ax.plot(range(len(col)), col, color='#FF5733')

        ax.set\_xticks(range(len(col)))

        ax.set\_xticklabels(['Column{}'.format(i+1) for i in range(len(col))])

        plt.show()

ui\_form.py

from PySide6.QtCore import (QCoreApplication, QDate, QDateTime, QLocale,

    QMetaObject, QObject, QPoint, QRect,

    QSize, QTime, QUrl, Qt)

from PySide6.QtGui import (QBrush, QColor, QConicalGradient, QCursor,

    QFont, QFontDatabase, QGradient, QIcon,

    QImage, QKeySequence, QLinearGradient, QPainter,

    QPalette, QPixmap, QRadialGradient, QTransform)

from PySide6.QtWidgets import (QApplication, QHeaderView, QPushButton, QSizePolicy,

    QTableView, QWidget)

class Ui\_Widget(object):

    def setupUi(self, Widget):

        if not Widget.objectName():

            Widget.setObjectName(u"Widget")

        Widget.resize(1005, 637)

        self.btn\_file = QPushButton(Widget)

        self.btn\_file.setObjectName(u"btn\_file")

        self.btn\_file.setGeometry(QRect(70, 40, 101, 31))

        self.tableView = QTableView(Widget)

        self.tableView.setObjectName(u"tableView")

        self.tableView.setGeometry(QRect(20, 90, 331, 421))

        self.tableView\_2 = QTableView(Widget)

        self.tableView\_2.setObjectName(u"tableView\_2")

        self.tableView\_2.setGeometry(QRect(360, 90, 421, 281))

        self.btn\_calc = QPushButton(Widget)

        self.btn\_calc.setObjectName(u"btn\_calc")

        self.btn\_calc.setGeometry(QRect(190, 40, 101, 31))

        self.btn\_bubble = QPushButton(Widget)

        self.btn\_bubble.setObjectName(u"btn\_bubble")

        self.btn\_bubble.setGeometry(QRect(300, 40, 101, 31))

        self.btn\_scatter\_matrix = QPushButton(Widget)

        self.btn\_scatter\_matrix.setObjectName(u"btn\_scatter\_matrix")

        self.btn\_scatter\_matrix.setGeometry(QRect(410, 40, 111, 31))

        self.btn\_par\_coord = QPushButton(Widget)

        self.btn\_par\_coord.setObjectName(u"btn\_par\_coord")

        self.btn\_par\_coord.setGeometry(QRect(530, 40, 151, 31))

        self.retranslateUi(Widget)

        QMetaObject.connectSlotsByName(Widget)

    # setupUi

    def retranslateUi(self, Widget):

        Widget.setWindowTitle(QCoreApplication.translate("Widget", u"Widget", None))

        self.btn\_file.setText(QCoreApplication.translate("Widget", u"File", None))

        self.btn\_calc.setText(QCoreApplication.translate("Widget", u"Calc", None))

        self.btn\_bubble.setText(QCoreApplication.translate("Widget", u"Bubble", None))

        self.btn\_scatter\_matrix.setText(QCoreApplication.translate("Widget", u"Scatter matrix", None))

        self.btn\_par\_coord.setText(QCoreApplication.translate("Widget", u"Parallel coordinates", None))