Matplotlib 及 Word 操作总结

AUTHOR: 彭玲 TIME: 2023/12/22

Matplotlib

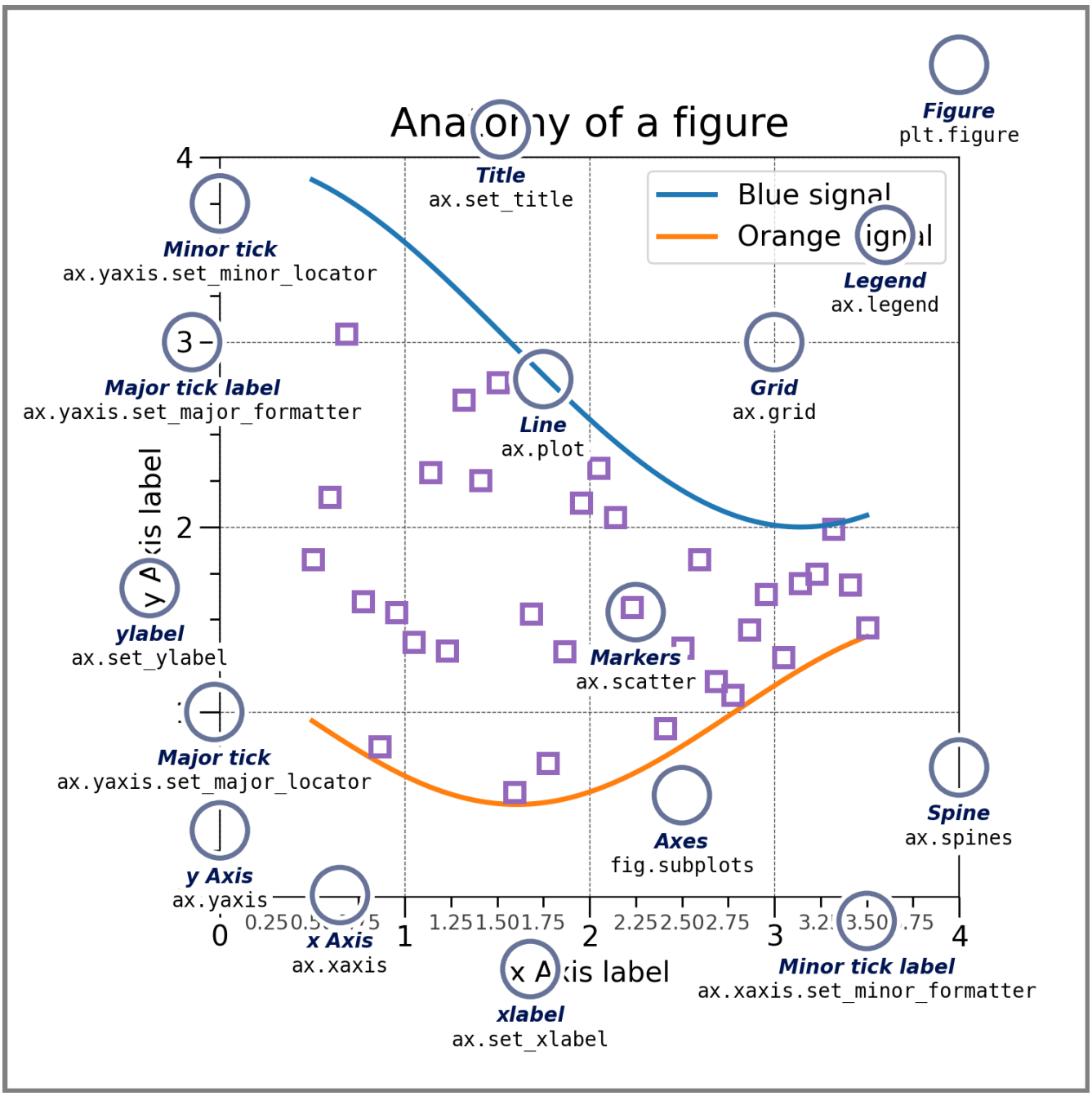
简介

Matplotlib 是 Python 最著名的 2D 绘图库，提供了丰富的数据绘图工具，主要用于绘制一些统计图形。Matplotlib 可用于 Python 脚本，Python 和 IPython shell，Jupyter Notebook，Web应用程序服务器和四个图形用户界面工具包。

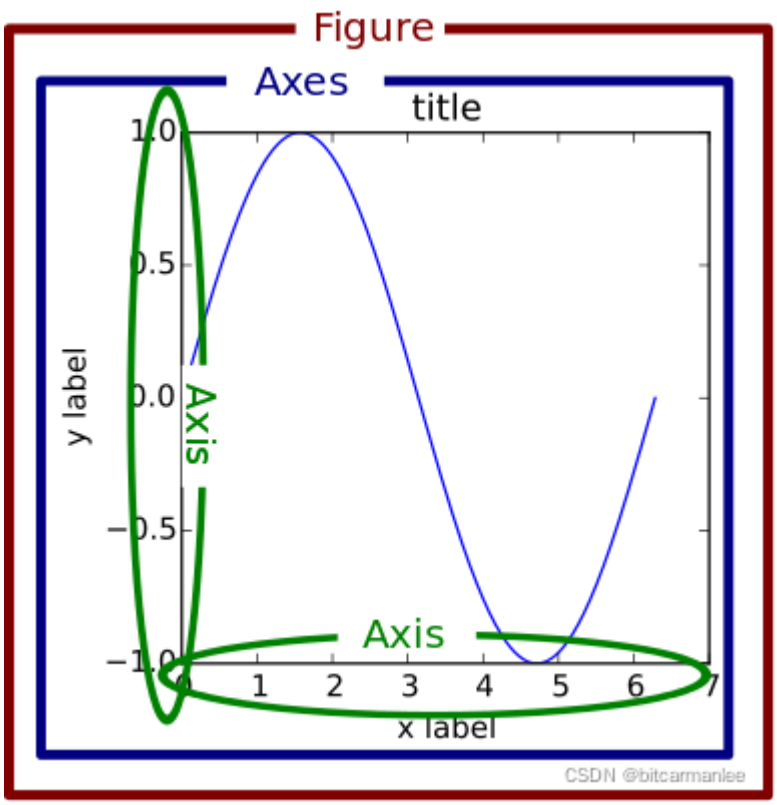
Matplotlib API 官方教程：https://matplotlib.org/stable/api/index

组件

在使用过程中，我们经常会发现 Matplotlib 里面包含各种元素，像 Figure，Axes，Axis 等等出现频率很高，而且实现的方式多种多样。开始阶段很可能会对 Matplotlib API 的使用有些迷茫，主要原因是对 Figure，Axes，Axis 等这些概念理解不清楚。要对 Matplotlib 理解清楚，熟练使用 Matplotlib API 进行绘图，需要从根本上理解清楚这些概念。首先，最最最重要的是认识 Matplotlib Figure 的组成部分，如下图所示（更多说明详见 官方有关 Parts of a Figure）：



重点看下 Figure，Axes，Axis 这三个概念：



Figure

Figure 是图中的红色区域，可以简单地认为就是整个画布。画图的第一件事就是创建一个`figure`，然后基于该`figure`做各种操作。

Axes 和 Axis

`Axes`这个名称很误导人，尤其容易与`Axis`混淆（Axes 与 Axis 不同，并**不是 Axis 的复数**形式）：

- `axis`是图中的绿色区域，是坐标轴部分。
- `axes`可以认为是一个`plot`，图中的蓝色部分。换句话说，`axes`才是真正的图片内容，包含`axis`、`title`、`x-label`和`y-label`。
- 一个`figure`里面可以包含多个`axes`，但是一个`axes`只能在一个`figure`中。

使用

接触 Matplotlib 是从项企流程效能分析项目开始的，这里把在项目中的使用做个总结分享，供后续使用参考。

关键代码

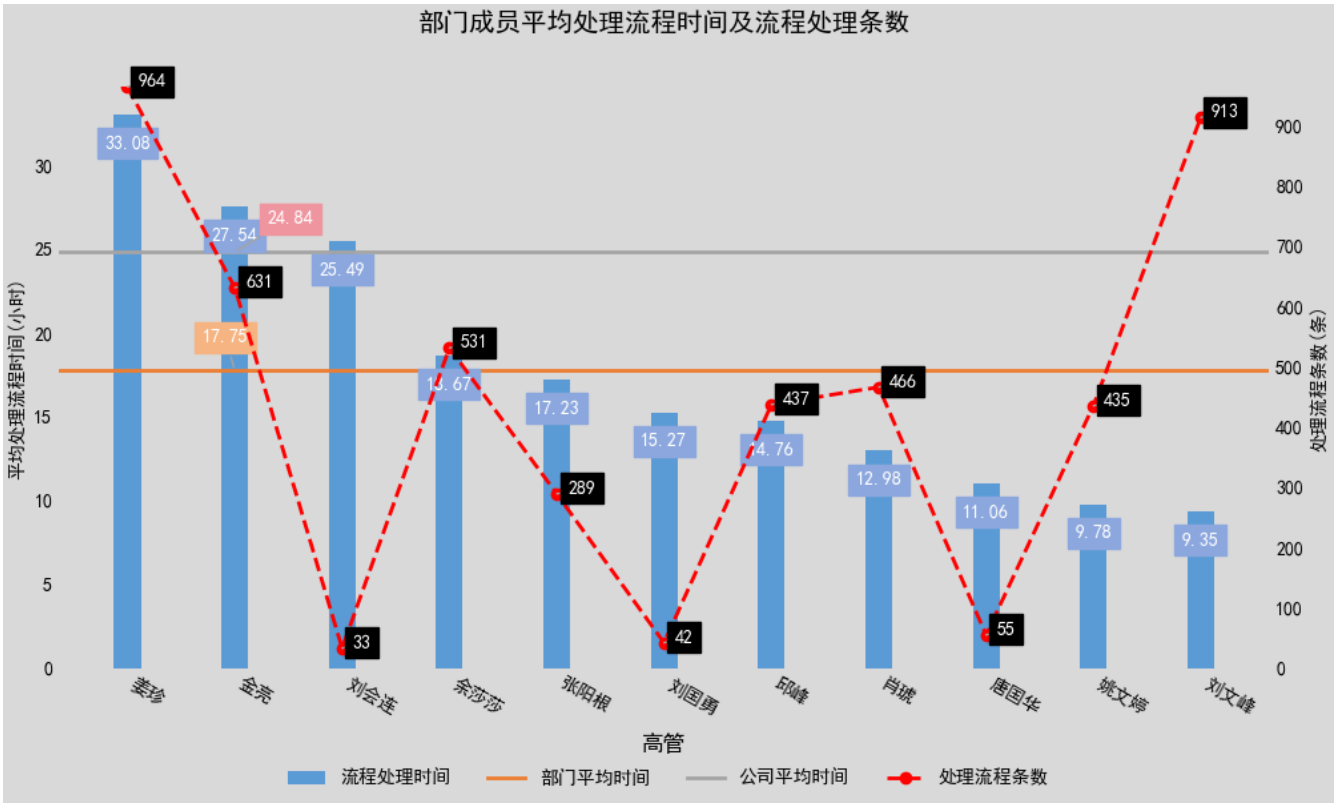
```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np

def create_twinx_chart(department, title, filename):
    plt.rcParams["font.sans-serif"] = ["SimHei"]  # 设置字体
    plt.rcParams["axes.unicode_minus"] = False  # 正常显示负号

    x = []
    y1 = []
```

```
    y2 = []
    for user in department['d_users']:
        x.append(user[0])  # 用户名
        y1.append(user[2])  # 用户处理流程耗时
        y2.append(user[1])  # 用户处理流程条数

    x_range = np.arange(len(x))

    bgcolor = '#d9d9d9'

    fig, ax1 = plt.subplots(figsize=(10, 6), layout='constrained', facecolor=bgcolor)
    ax1.set_ylabel('平均处理流程时间(小时)')
    y1_bar = ax1.bar(x_range, y1, width=0.25, label='流程处理时间', color=(91 / 255, 155 / 255, 213 / 255))

    ax1.set_xticks(x_range, x)
    # 设置x轴标签旋转角度
    plt.xticks(rotation=-30, ha='left')

    ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

    y2_line, = ax2.plot(x_range, y2, label='处理流程条数', color='#ff0000', marker='o', linestyle='dashed', linewidth=2)
    ax2.set_ylabel('处理流程条数(条)')
    ax2.set_ylim(0, max(y2) + 1)
    ax2.yaxis.set_major_locator(ticker.MaxNLocator(integer=True))  # 设置整数刻度

    annotate_kwargs = {'color': '#ffffff', 'arrowprops': dict(color='#a5a5a5', arrowstyle='-')}

    # y2_line: 添加数据标签
    # UserWarning: constrained_layout not applied because axes sizes collapsed to zero.
    for x, y in zip(x_range, y2):
        xtext_offset = 0.01 if len(x_range) == 1 else 0.1
        ax2.annotate(f"{y}", xy=(x, y), xytext=(x + xtext_offset, y + 0.1), backgroundcolor='#000000', **annotate_kwargs)

    # 绘制均值线: 相对于左侧坐标轴"平均处理流程时间(小时)"绘制
    hline_kwargs = {'linestyle': 'solid', 'linewidth': 2}
    axhline1 = ax1.axhline(department['d_elapse'], label='部门平均时间', color='#ed7d31', **hline_kwargs)
    axhline2 = ax1.axhline(procinst_duration_by_company, label='公司平均时间', color='#a5a5a5', **hline_kwargs)

    # 获取 y1 轴两个刻度之间的距离
    y1_ticks = ax1.get_yticks()
    y1_ticks_distance = y1_ticks[1] - y1_ticks[0] if len(y1_ticks) > 1 else y1_ticks[0]

    # y1_bar: 添加数据标签
    bar_label_kwargs = {'color': '#ffffff', 'backgroundcolor': '#8ba7de'}
    y1_threshold = y1_ticks_distance / 2
```

```
    y1_bar_large = [v if v > y1_threshold else '' for v in y1]
    y1_bar_small = [v if v <= y1_threshold else '' for v in y1]
    ax1.bar_label(y1_bar, y1_bar_large, padding=-20, **bar_label_kwargs)
    ax1.bar_label(y1_bar, y1_bar_small, padding=3, **bar_label_kwargs)

    # 均值线：添加数据标签
    x_pos = 0 if len(x_range) == 1 else 1
    xtext_benchmark = 0 if len(x_range) == 1 else 1
    xtext_offset = 0.03 if len(x_range) == 1 else 0.3
    ytext_offset = y1_ticks_distance / 3
    ax1.annotate(f"{department['d_elapse']}", xy=(x_pos, department['d_elapse']),
                 xytext=(xtext_benchmark - xtext_offset, department['d_elapse'] +
ytext_offset),
                 backgroundcolor='#f5b482', **annotate_kwargs)
    ax1.annotate(f"{procinst_duration_by_company}", xy=(x_pos,
procinst_duration_by_company),
                 xytext=(xtext_benchmark + xtext_offset, procinst_duration_by_company
+ ytext_offset),
                 backgroundcolor='#ef95a0', **annotate_kwargs)

    plt.title(title, fontsize=14, pad=30)  # pad: 标题与坐标轴顶部的偏移量
    fig.legend(handles=[y1_bar, axhline1, axhline2, y2_line], loc='outside lower
center', ncols=4, facecolor=bgcolor, edgecolor=bgcolor)

    ax1.set_xlabel(department['d_name'], fontsize=12, labelpad=8)  # 在图形下方显示部门
名称
    ax1.set_facecolor(bgcolor)  # 设置绘图区域的背景色
    # 去掉绘图区域的边框线和坐标轴刻度线
    axes = [ax1, ax2]
    for ax in axes:
        ax.spines['top'].set_visible(False)
        ax.spines['bottom'].set_visible(False)
        ax.spines['left'].set_visible(False)
        ax.spines['right'].set_visible(False)
        ax.tick_params(axis='both', which='both', length=0)

    # 设置绘图区与图片边缘的距离
    # plt.subplots_adjust(left=0.08, right=0.94, top=0.9, bottom=0.24)

    # 保存图形
    plt.savefig('{}/{}'.format(REPORT_DIR, filename))
    plt.close()
```

图形效果

上面的代码功能：绘制一个双 y 坐标轴的 Chart，`流程处理时间` 用柱状图表示，`流程处理条数` 用曲线图表示，`部门平均时间` 和 `公司平均时间` 用水平线表示，效果如下。



代码说明

图形中如果有中文的话，需要进行以下设置，否则显示乱码（构建时需要提供`SimHei.ttf`字体文件，防止系统中没有该字体）：

```
    plt.rcParams["font.sans-serif"] = ["SimHei"]  # 设置字体
    plt.rcParams["axes.unicode_minus"] = False  # 正常显示负号
```

图形布局：使用`constrained`布局方式，使图形能够最大化显示在 Figure 中，省得各种边距的手动调整。

可以使用`plt.subplots_adjust()`方法设置绘图区与图片边缘的距离，但使用该方法时不能使用`constrained`布局，否则`subplots_adjust()`方法无效。

```
    fig, ax1 = plt.subplots(figsize=(10, 6), layout='constrained', facecolor=bgcolor)

    # 设置绘图区与图片边缘的距离
    # plt.subplots_adjust(left=0.08, right=0.94, top=0.9, bottom=0.24)
```

绘制右侧 y 坐标轴：

```
    ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
```

设置整数刻度：右侧 y 坐标轴为`处理流程条数`，为整数值，要显示整数刻度需要特别设置（默认会显示小数）。

```
    ax2.yaxis.set_major_locator(ticker.MaxNLocator(integer=True))  # 设置整数刻度
```

绘制均值线：由于`部门平均时间`和`公司平均时间`作为每个人`平均处理流程时间`的参考基线，因此，水平线需要基于左侧 y 轴绘制，使用`ax1`这个 Axes 对象绘制。

```
    # 绘制均值线：相对于左侧坐标轴"平均处理流程时间(小时)"绘制
    hline_kwargs = {'linestyle': 'solid', 'linewidth': 2}
    axhline1 = ax1.axhline(department['d_elapse'], label='部门平均时间',
color='#ed7d31', **hline_kwargs)
    axhline2 = ax1.axhline(procinst_duration_by_company, label='公司平均时间',
color='#a5a5a5', **hline_kwargs)
```

设置图形`标题`与`坐标轴顶部`的距离：通过`title()`方法的`pad`属性进行设置。

```
    plt.title(title, fontsize=14, pad=30)  # pad: 标题与坐标轴顶部的偏移量
```

利用 x 轴标签默认在图形下方显示这一特点，设置`部门名称`（这里为`高管`），设置绘图区域背景色等：这里使用的是`ax1`对象进行操作，使用`ax2`对象操作无效。

```
    ax1.set_xlabel(department['d_name'], fontsize=12, labelpad=8)  # 在图形下方显示部门名
称
    ax1.set_facecolor(bgcolor)  # 设置绘图区域的背景色
```

添加数据标签：通常通过 Axes 的`annotate()`方法给数据点添加标签说明，柱状图可以通过`bar_label()`方法为每个柱状图添加数据标签。

在项企流程效能分析中，有的成员流程处理时间很小，有的时间很大，为了能够自适应地显示数据标签，以下代码做了特别处理：设置一个时间阈值，对于超过这个阈值的柱状图，其数据标签显示在柱状图顶部的`下方`；对于低于这个阈值的柱状图，其数据标签显示在柱状图顶部的`上方`。

```
    # 获取 y1 轴两个刻度之间的距离
    y1_ticks = ax1.get_yticks()
    y1_ticks_distance = y1_ticks[1] - y1_ticks[0] if len(y1_ticks) > 1 else
y1_ticks[0]

    # y1_bar: 添加数据标签
    bar_label_kwargs = {'color': '#ffffff', 'backgroundcolor': '#8ba7de'}
    y1_threshold = y1_ticks_distance / 2
    y1_bar_large = [v if v > y1_threshold else '' for v in y1]
    y1_bar_small = [v if v <= y1_threshold else '' for v in y1]
    ax1.bar_label(y1_bar, y1_bar_large, padding=-20, **bar_label_kwargs)
    ax1.bar_label(y1_bar, y1_bar_small, padding=3, **bar_label_kwargs)
```

设置图例：默认情况下，图例会有自己的显示顺序，如果要指定各图例的显示顺序，需要通过`handles`属性进行设置。

```
fig.legend(handles=[y1_bar, axhline1, axhline2, y2_line], loc='outside lower center',
ncols=4, facecolor=bgcolor, edgecolor=bgcolor)
```

注意：`ax2.plot()`返回的是`Line2D`列表，作为`handles`参数，需要获取`Line2D`对象（通过下面的语句解构 list 获取`Line2D`对象）。

```
    y2_line, = ax2.plot(x_range, y2, label='处理流程条数', color='#ff0000', marker='o',
linestyle='dashed', linewidth=2)
```

默认情况下，绘图区域会显示边框线和坐标轴刻度线，要去掉边框线和坐标轴刻度线，需要对两个`Axes`对象进行设置：

```
    # 去掉绘图区域的边框线和坐标轴刻度线
    axes = [ax1, ax2]
    for ax in axes:
        ax.spines['top'].set_visible(False)
        ax.spines['bottom'].set_visible(False)
        ax.spines['left'].set_visible(False)
        ax.spines['right'].set_visible(False)
        ax.tick_params(axis='both', which='both', length=0)
```

小结

至此，通过以上内容，我们对 Matplotlib 的基本概念及使用有了初步的认识，如果要熟练使用，还是需要实际编码去加深理解的。小结下使用上的几点心得：

- 要使用 Matplotlib API 必须先理解 Matplotlib 的基本概念（详见上面的`组件`一节）。
- Matplotlib API 方法还是比较多的，每个方法又有很多参数，具体使用哪个 API 的什么参数或属性，这个还是需要通过实操看效果的，毕竟很多属性的说明不能很明确地指导我们，只能多操作，多积累经验吧...
- 可以借助 Chrome 或者 Chat 帮助我们快速找到答案，当然还是要实操去验证，毕竟有时候给的答案是不准确的。

Word

简介

`python-docx`是一个利用 Python 来读写 Word 文件的第三方库。是一个用于创建和更新 Microsoft Word (.docx) 文件的库，提供全套的 Word 操作，是最常用的 Word 工具。

官方文档：https://python-docx.readthedocs.io/en/latest/

基本概念

- Document：是一个 Word 文档对象，打开不同的 Word 文档，就会有不同的 Document 对象，相互之间没有影响。
- Paragraph：是段落，一个 Word 文档由多个段落组成，当在文档中输入一个回车键，就会成为新的段落。
- Run：表示一个节段，每个段落由多个 节段 组成，一个段落中具有相同样式的连续文本，组成一个节段，所以一个 段落 对象有个 Run 列表。

使用

在安装时使用的名字是`python-docx`，但是在导入时是另一个名字`docx`。

关键代码

同样，下面以项企流程效能分析项目中的应用对`python-docx`的使用进行说明，供后续使用参考。

```python
from docx import Document
from docx.shared import Inches, Cm, Pt, RGBColor
from docx.enum.text import WD_PARAGRAPH_ALIGNMENT, WD_LINE_SPACING
from docx.enum.table import WD_TABLE_ALIGNMENT, WD_CELL_VERTICAL_ALIGNMENT
from docx.oxml.ns import qn, nsdecls
from docx.oxml import parse_xml

def generate_word_report():
    # 创建一个新的word文档
    doc = Document()
    set_global_normal_style(doc)

    # 添加标题
    h = doc.add_heading(level=0)
    r = h.add_run('{}月项企流程效能分析结果公示'.format(last_month))
    set_title_format(h, r)

    # 添加段落
    p_first = doc.add_paragraph(
        '为了提升公司整体工作效率，确保跨部门工作高效推进，充分发挥我司自研平台的优势，现基于项企系统数据对我司项企平台的使用数据进行分析，'
        '进而通过量化数据发现我司办公流程流转中的问题，'
        '现将{}年{}月公司员工处理流程平均响应时长（响应时长为从流程流转到当前节点到当前节点处理完成的耗时，单位：小时）'
        '以及处理流程条数情况结果进行公示如下：'.format(last_year, last_month))
    set_paragraph_format(p_first)

    # 一、公司各部门横向比较情况
    add_chapter_1(doc)

    # 二、各部门成员详情
    add_chapter_2(doc)

    p_last = doc.add_paragraph(
        '针对以上数据，请各部门负责人认真对待并针对部门情况进行分析。'
        '现项企业流程效能分析是公司管理工具逐步完善尝试和摸索的阶段，评估维度、呈现形式均在调整优化中，'
        '欢迎大家对效能评估维提出建设意见，共同促进项企流程效能分析不断完善，进而提升公司整体效率！')
    set_paragraph_format(p_last)

    # 保存文档
    report_file_name = '项企流程效能分析结果公示-{}年{}月.docx'.format(last_year, last_month)
    report_file = f'{REPORT_DIR}/{report_file_name}'
    doc.save(report_file)
```

word 效果

11 月项企流程效能分析结果公示

为了提升公司整体工作效率，确保跨部门工作高效推进，充分发挥我司自研平台的优势，现基于项企系统数据对我司项企平台的使用数据进行分析，进而通过量化数据发现我司办公流程流转中的问题，现将 2023 年 11 月公司员工处理流程平均响应时长（响应时长为从流程流转到当前节点到当前节点处理完成的耗时，单位：小时）以及处理流程条数情况结果进行公示如下：

一、公司各部门横向比较情况

（一）一览表

2023 年 11 月，公司共计 189 人使用项企平台，个人处理流程平均耗时 24.84 小时/流程，平均单一流程处理耗时高于公司平均数的有 7 个部门，63 人。个人处理流程平均条数为 90 条，个人处理流程平均条数高于公司平均数的有 8 个部门，46 人。

序号	部门	人均处理流程条数	平均单条流程处理时间
1	高管	436	17.75
2	人力资源部	278	24.76
3	计划部	219	18.35
4	商务部	210	19.66
5	法务风控部	140	16.77
6	采购部	128	19.84
7	行政管理部	117	26.49
8	财务管理部	116	20.86
9	营销第九独立团	79	12.18
10	品检部	74	39.03
11	营销第五独立团	72	20.34
12	技术方案部	70	24.47
13	营销第二作战师	69	28.42
14	客户服务管理部	55	23.1

代码说明

设置全文中英文字体，段前段后距离，页边距：

```python
def set_global_normal_style(doc):
    style = doc.styles['Normal']
    style.font.name = 'Times New Roman'  # 英文、数字字体
    style._element.rPr.rFonts.set(qn('w:eastAsia'), u'宋体')  # 中文字体
    style.font.size = Pt(14)  # 四号
    style.paragraph_format.space_before = 0  # 段前
    style.paragraph_format.space_after = 0  # 段后
    style.paragraph_format.line_spacing_rule = WD_LINE_SPACING.SINGLE  # 单倍行距
    # 设置页边距
    section = doc.sections[0]
```

```
    section.top_margin = Cm(1.27)
    section.right_margin = Cm(0.5)
    section.bottom_margin = Cm(1.27)
    section.left_margin = Cm(0.5)
```

设置标题居中，字体加粗等：

```
def set_title_format(h, r):
    h.paragraph_format.space_before = Pt(6)  # 段前0.5行
    h.paragraph_format.space_after = Pt(6)  # 段后0.5行
    h.paragraph_format.line_spacing = 1.5  # 1.5倍行距
    h.alignment = WD_PARAGRAPH_ALIGNMENT.CENTER
    r.font.name = u'Times New Roman'
    r._element.rPr.rFonts.set(qn('w:eastAsia'), u'宋体')
    r.font.size = Pt(18)  # 小二
    r.font.color.rgb = RGBColor(0, 0, 0)
    r.bold = True
```

设置段落：首行缩进等。

```
def set_paragraph_space(p):
    # 设置间距
    p.paragraph_format.space_before = Pt(6)  # 段前
    p.paragraph_format.space_after = Pt(6)  # 段后
    p.paragraph_format.line_spacing = 1.5  # 设置段落行距：1.5倍行距


def set_paragraph_format(p):
    set_paragraph_space(p)
    # 设置首行缩进2个字符
    p.paragraph_format.first_line_indent = Inches(0.3)
```

文档中插入表格：

```
def insert_table(doc, table_headers, col_width_dict, table_data, table_style):
    # 在文档中添加一个空表格
    table = doc.add_table(rows=0, cols=len(table_headers), style='Table Grid')

    # 添加表头
    heading_cells = table.add_row().cells
    for i, header in enumerate(table_headers):
        heading_cells[i].text = header
        heading_cells[i].paragraphs[0].runs[0].bold = True  # 字体加粗

    # 循环添加数据
    avg_row_added = False
    for i, row_data in enumerate(table_data):
        if table_style in ['style2', 'style3']:
            avg_value = {'style2': procinst_duration_by_company, 'style3':
procinst_count_by_company}
            if not avg_row_added and row_data[1] < avg_value[table_style]:
```

```
                # 添加“平均”值行
                row = table.add_row()
                row_cells = row.cells
                row_cells[0].merge(row_cells[1]).text = '平均'  # 合并单元格
                row_cells[2].text = str(avg_value[table_style])
                set_table_row_shading_color(row, 'ff0000')
                row_cells[0].paragraphs[0].runs[0].bold = True  # 字体加粗
                row_cells[2].paragraphs[0].runs[0].bold = True
                avg_row_added = True

        row = table.add_row()
        row_cells = row.cells

        row_cells[0].text = str(i + 1)
        for j, cell_value in enumerate(row_data):
            col_index = j + 1
            row_cells[col_index].text = str(cell_value)

        if table_style == 'style1':
            if row_data[1] >= procinst_count_by_company:
                set_table_cell_shading_color(row_cells[2], '#fedb61')
            if row_data[2] >= procinst_duration_by_company:
                set_table_cell_shading_color(row_cells[3], '#ff0000')
        elif table_style == 'style2' and row_data[1] >= procinst_duration_by_company:
            set_table_row_shading_color(row, 'fedb61')

    if table_style == 'style1':
        # 统计值：平均值
        row_cells = table.add_row().cells
        avg_data = [len(table_data) + 1, '公司平均', procinst_count_by_company,
procinst_duration_by_company]
        for i, cell_value in enumerate(avg_data):
            row_cells[i].text = str(cell_value)
            row_cells[i].paragraphs[0].runs[0].bold = True  # 字体加粗

    # 设置表格垂直对齐方式
    table.alignment = WD_TABLE_ALIGNMENT.CENTER

    # 设置单元格样式
    for row in table.rows:
        for cell in row.cells:
            cell.vertical_alignment = WD_CELL_VERTICAL_ALIGNMENT.CENTER  # 垂直居中
            cell.paragraphs[0].alignment = WD_PARAGRAPH_ALIGNMENT.CENTER  # 水平居中
        for col_index, col_width in col_width_dict.items():  # 设置列宽
            row.cells[col_index].width = Cm(col_width)
        row.height = Cm(0.7)  # 设置行高
```

设置表格行底纹：

```python
def set_table_row_shading_color(table_row, color):
    shading_dict = {}
    for i, cell in enumerate(table_row.cells):
        shading_dict['shading_elm_' + str(i)] = parse_xml(r'<w:shd {} w:fill="{bgColor}"/>'.format(nsdecls('w'), bgColor=color))
        cell._tc.get_or_add_tcPr().append(shading_dict['shading_elm_' + str(i)])
```

设置单元格底纹：

```python
def set_table_cell_shading_color(table_cell, color):
    shading_elm = parse_xml(r'<w:shd {} w:fill="{bgColor}"/>'.format(nsdecls('w'), bgColor=color))
    table_cell._tc.get_or_add_tcPr().append(shading_elm)
```

小结

相对于 Matplotlib，`python-docx` 官方文档在使用上比较费劲，项目应用中很多功能（比如上面的 `中英文字体` 设置、表格的可选样式、`单元格底纹` 设置等）无法通过官方文档直观地获取或者说很难找到使用说明。相较于官方文档，通过 Chrome 或者 Chat 更容易找到答案。