数据可视化实验报告

计算机科学与技术学院

班　　级：大数据 2101 班

学　　号：U202115578

姓　　名：罗理恒

指导教师：何云峰

完成日期：2024 年 1 月 10 日

实验报告及设计评分细则

评 分 项 目	满分	得分	备注	
文档格式（段落、行间距、缩进、图表、编号等）	15			实 验 报 告 总分
实验方案设计	10			
实验过程	50			
遇到的问题及处理	10			
设计方案存在的不足	5			
心得（含思政）	5			
意见和建议	5			
可视化作品	100			
教师签名			日　　期	

备注：实验过程将从可视化作品的完成度、是否讲述好数据的故事、作品的复杂度等方面进行综合评分。

实验课程总分=可视化*0.6+实验报告*0.4

目　录

1 实验概述

1.1 实验名称

设计一个数据可视化作品，讲述数据的故事。

1.2 实验目的

（1）了解数据可视化方案的设计和实现方法；

（2）利用 python 的数据可视化工具库（matplotlib 库、pyecharts 库）进行可视化方案的实现。

1.3 实验环境

软件：python。

可用库：数据处理库 NumPy、pandas；可视化库 matplotlib 库、pyecharts 库等。

可视化视图：pyecharts 中提供了 Bar(柱状图/条形图)；Bar3D(3D 柱状图)；Boxplot(箱形图)；EffectScatter(涟漪散点图)；Funnel (漏斗图)；Gauge(仪表盘)；Geo(地理坐标系)；Graph(关系图)；HeatMap(热力图)；Kline(K 线图)；Line(折线/面积图)；Line3D(3D 折线图)；Liquid(水球图)；Map(地图)；Parallel (平行坐标系；Pie(饼图)；Polar(极坐标系)；Radar(雷达图)；Sankey(桑基图)；Scatter(散点图等，可以根据需要进行选择。

1.4 实验内容

（1）在备选的 4 个数据集中任意选择其中一个数据集，对数据集中的数据进行数据分析和预处理，设计合理的可视化方案。

（2）利用 python 的可视化工具库实现可视化方案。

（3）完成实验报告，讲述数据的故事。

1.5 实验要求

（1）对数据集中的数据进行分析和整理，提取出其中的有效信息，完成数据的预处理；

（2）设计合理的可视化视图，选择合适的视觉通道，表达数据的某个特性；

（3）完成不少于 6 个可视化视图，较为完整地描述数据集。

2 总体设计

2.1 总体设计思路

我选择的数据集是麦当劳的评论数据，数据集较为原始，数据来源是阿里云。

首先对数据集进行简要的分析。该数据集总共有 33369 条评论数据，特征包括评论号（review_id），店铺名称（store_name），店铺类型（category），店铺地址（store_name），纬度（latitude），经度（longitude），评价数（rating_count），评价内容（review），评价分数（rating）。可以看出，该数据集的主要分为地理型数据、数值型数据和文字型数据。

接着分析数据集数据的特性，可以看出店铺名称和店铺类型是相同的，均为 Mcdonald。数据集为每家麦当劳提供了经纬度进行定位，每一家 Mcdonald 商店都有上千条评论以及用户对它的文字评价。

基于对数据集的理解与分析，我将从数据处理开始，完成数据清洗等操作，生成用于可视化的精细数据，接着确定和完成可视化图表的绘制，最后设计用户的交互方式。

2.2 总体设计框架

可视化流程的总体设计框架如图 2.1 所示（省略了数据采集和用户感知部分）：

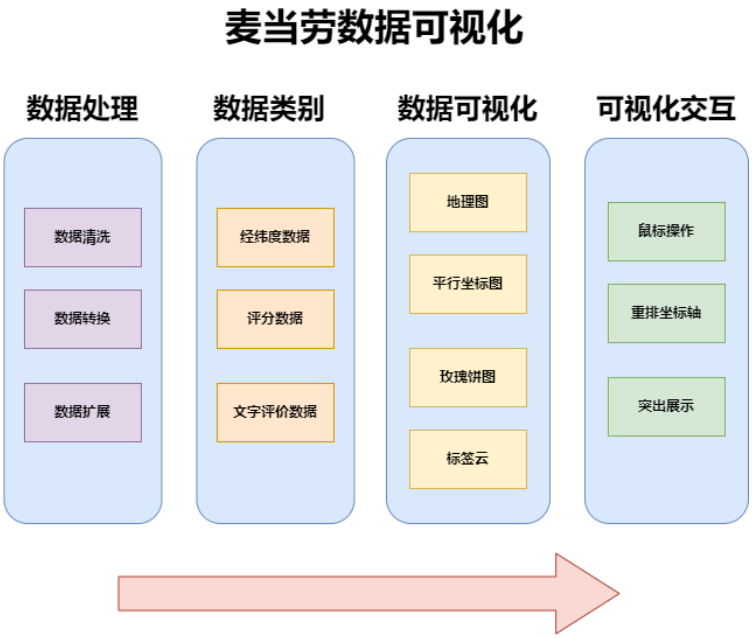

图 2.1 麦当劳可视化总体设计图

基于数据的特性，我准备从以下几个方面对数据的特性进行可视化分析：

1. 对数据集进行预处理，包括数据清洗，数据转换以及数据扩展，使得数据更易可视化。

2. 对店铺经纬度进行可视化，分析店铺的地理分布特征

3. 对店铺的用户平均评分进行可视化，分析店铺的服务质量

4. 对用户评论内容进行可视化，分析用户对麦当劳的评价

确定了可视化对象后，接下来对数据进行预处理，使得数据能够适用于可视化代码要求，具体的步骤将在 3.1 节中详细展开。

接下来选定可视化图库，我选用的可视化库为 pyecharts，原因是 pyecharts 的用户交互设计非常方便。

为了展示店铺的地理分布特征，我选择使用地理图标注店铺的位置；由于每个店铺的特征较多，我选择了使用平行坐标图去表示每家店铺的对比；为了可视化店铺的评分分布情况以及地理分布情况，我选择使用饼图进行统计；最后为了可视化用户的评价内容，我采用标签云图呈现用户对麦当劳的整体评价以及对高评分店与低评分店的评价。

最后设计可视化图表的用户交互方式。由于使用了平行坐标图，我可以选择“选中”操作以突出展示某些关键数据，“重排坐标轴”操作以展现相邻坐标轴之间的联系，“过滤、刷取”操作达到筛选数据的目的。至于其他的可视化视图，交互方式主要以鼠标操作为主，比如鼠标点击展示相关信息等。

3 实验过程

3.1 数据预处理

数据预处理分为三个部分，数据清洗，数据转换和数据扩展。

（1）数据清洗

在实验过程中，我发现该数据集存在以下几个问题：

一是评论部分存在乱码，分析应该是引号无法正常显示，但是这对整体的可视化没有影响，删去即可。

二是在统计店铺总数时，发现同一店铺的 rating_count 数据之间存在差异，虽然差异很小，但是影响了代码结果，因此需要调整到相同的值，在处理过程中我保留了更小的值，更具体地，在清洗之前通过经纬度和评论总数进行统计时总共有 53 家店铺，而在清洗过后仅剩 39 家店铺，这说明数据清洗对本数据集意义重大。

（2）数据转换

需要转换的数据包括 review_time 和 rating 数据项，因为数据项中带有英文，但是数据本身属于数值型数据，因此需要将其从 string 类型转换成 int 类型，比如将“3 months ago”转换成 90（此处假定一个月有 30 天），将“3 stars”转换成 3。

（3）数据扩展

由于原数据集所涵盖的信息量较少，因此我通过已知的数据项扩展了部分所需的新数据，具体如下：

一是通过店铺所在的经纬度坐标，扩展出每个店铺所在的州，比如（30.4607176，-97.7928744）在德克萨斯州等。这部分是通过 reverse_geocoder 库实现的转化。

二是每个店铺的平均用户评分，由于每条数据代表一个用户对一个店铺的评价和评分，所以需要通过聚合计算出平均评分。

三是每个店铺的档次，根据平均评分，我将所有店铺分为了 5 类，分别是 Poor, Fair, Normal, Good, Excellent，这样能够将同档次的店铺进行集中分析

3.2 地理视图

（1） 设计思路及设计过程

本图主要用来研究麦当劳店铺的店铺特征，并通过颜色，形状等视觉通道展

现出店铺的平均评分（或者档次）信息，同时通过鼠标悬停点击可以和地理图进行交互操作。

地理图设计如图 3.1 所示。

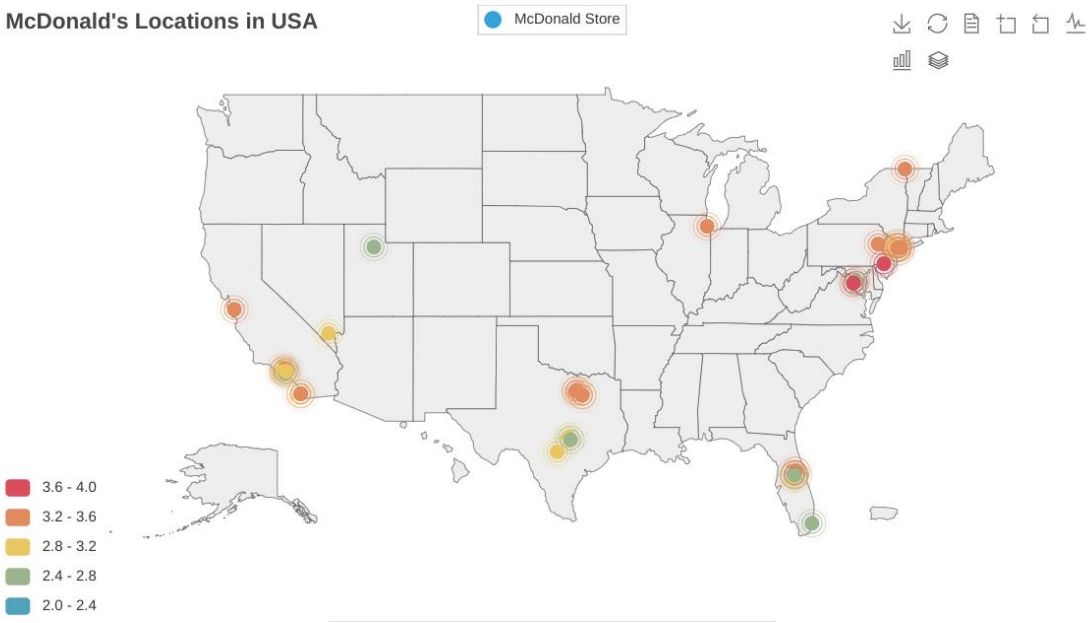

图 3.2 地理视图

（2） 交互设计

本图的交互设计主要有三点，一是通过鼠标悬停到某个州可以显示对应的州名，比如 Washington，California 等；二是鼠标悬停到某个商铺时会显示这家商铺的各项数据，比如平均评分，评价总数，经纬度等；三是通过鼠标滚轮可以放大地图中某一地区的分布情况。

（3）视图分析

首先，本地理图展现了数据集中麦当劳店铺的地理分布特征，结合后续的统计可以看出，本数据集中统计的店铺大多位于经济发达的地区，比如湾区的硅谷、首都华盛顿、纽约等，且由于数据集较少，店铺的地理分布不具有统计学上的特征。

其次，我通过颜色这个视觉通道标识了每个店铺的评价评分情况，具体的对应关系如图例所示，比如红色对应超级好评的店铺。

3.3 平行坐标视图

（1）设计思路及设计过程

本图主要用来比较每家店铺之间的多维数据，我选择的维度分别是用户所处在的州，店铺的评论总数，店铺的平均评分。同时我通过颜色区分了不同平均评分类别的店铺，以及设计了多种交互方式。

平行坐标视图如图 3.2 所示。

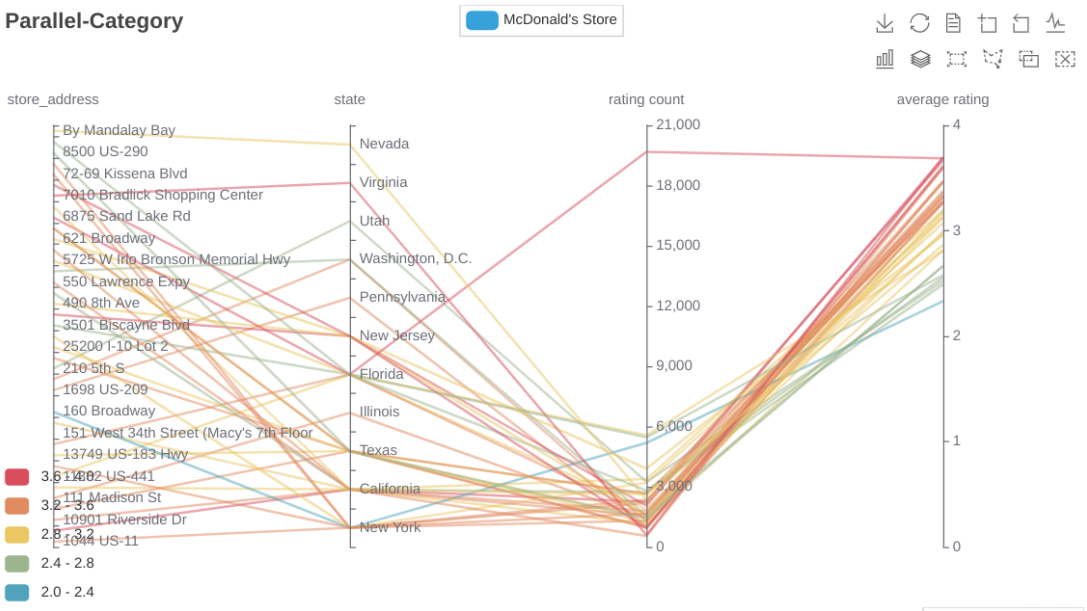

图 3.2 平行坐标视图

（2）交互设计

平行坐标视图的交互主要有以下四点。一是可以选中部分或者过来部分数据进行突出显示，比如图 3.3；二是重排坐标轴，可以通过鼠标拖动坐标轴实现位置的移动，从而比较相邻坐标之间的关系，图 3.2 展示的是一个比较合理的坐标轴分布；三是通过筛选数据达到部分数据的突出显示，比如可以只显示评分在 3.6 和 4.0 区间内的店铺数据；四是鼠标悬停在某项数据时可以展示店铺的经纬度信息等数据。

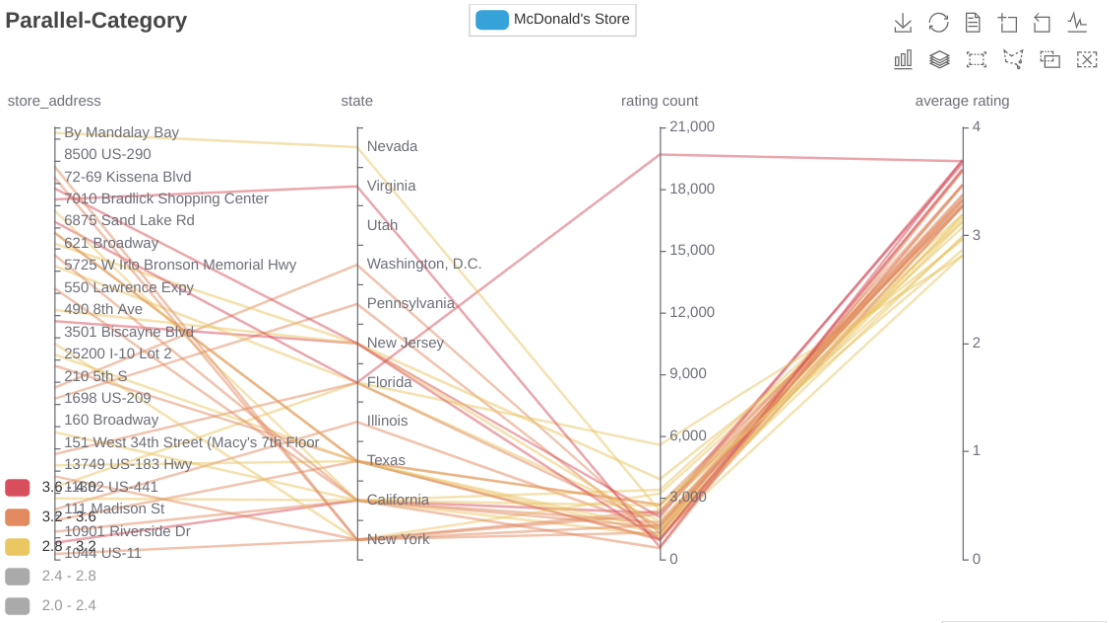

图 3.3 突出显示部分数据

（3）视图分析

平行坐标视图首先展示了每家店铺所在的州，其实这里可以进一步优化，将同一个州的店铺放在一起展示，这样能够比较州内店铺之间的差异。从评分总数我们可以看出，仅有一家店铺独树一帜，评分数超过了 20000，其余的店铺都处在 0-6000 的区间内。同时观察平行坐标视图可以发现，评分总数在一定程度上反映了这家店铺的用户活跃度以及人流量，但是和店铺的平均评分没有显著地联系，这也很符合麦当劳普通快餐的属性。

3.4 饼视图

（1）设计思路及设计过程

本视图主要主要展示数据集中两项特征的构成特征，一是店铺所在州的分布，二是店铺的评分类别（5 类）。采用饼状图可以直观地展示数据占比，使观众很容易理解数据之间的比例关系。为了加大不同类别之间的差异，我采用了玫瑰图，通过面积这个视觉通道进一步展现数据之间的比例关系。

饼视图如图 3.4，3.5 所示。

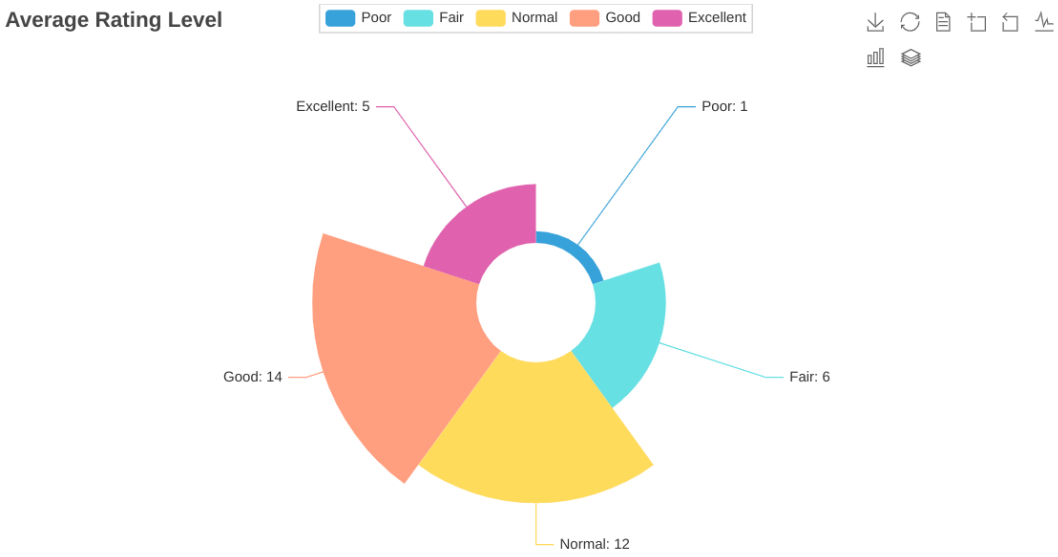

图 3.4 店铺评分类别

（2）交互设计

饼图的交互为鼠标悬停某个比例块时，会展示该块的数据信息，比如所占比例，对应类别等，同时该块会放大以提升视觉显示效果。

（3）视图分析

首先分析店铺评分类别，在 39 家统计店铺中，代表中间评分的 Good 和 Normal 类占比达到了 67%，而极好评和差评占比综合不超过 10%，这说明大部分客户对麦当劳的评价都比较客观且适中：能吃，会吃，但是不推荐吃（个人观点）。

其次分析店铺的州级分布，尽管在地理视图中已经对此进行过初步的观察，但是饼图更加直观地展示出了数据集中的店铺主要分布在经济发达或者人口众多的地区内。

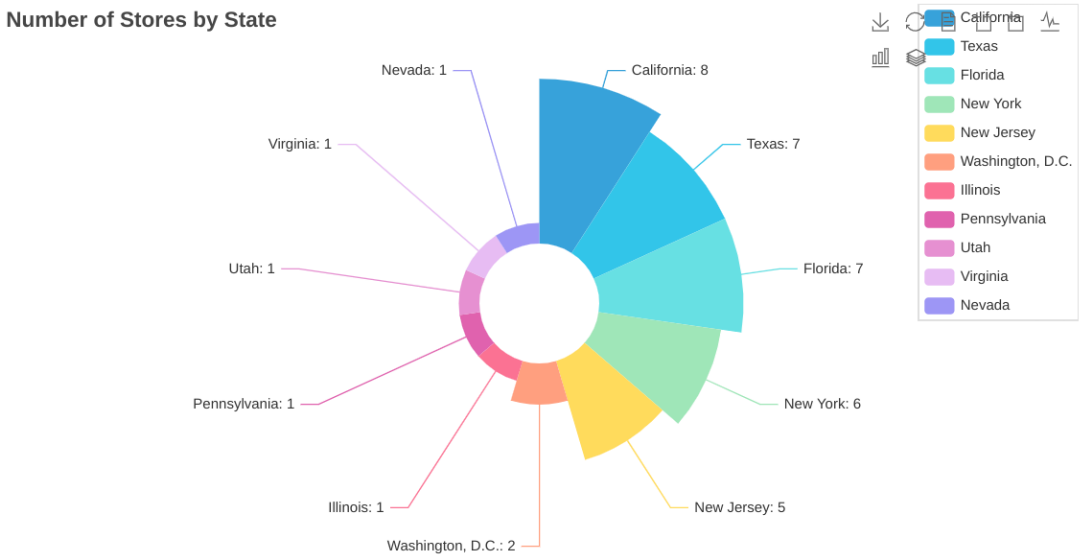

图 3.5 店铺的州分布

3.5 标签云视图

（1）设计思路及设计过程

本部分是该数据集的重点分析部分，通过标签云展现店铺客户对该店铺的评价。我将根据自己的思维路线展示标签云视图的设计思路。

首先我使用简单的标签云统计每个词在用户评论中出现的次数，得到了图3.6，可以看出，该图中主要是介词，副词等无用的词汇。

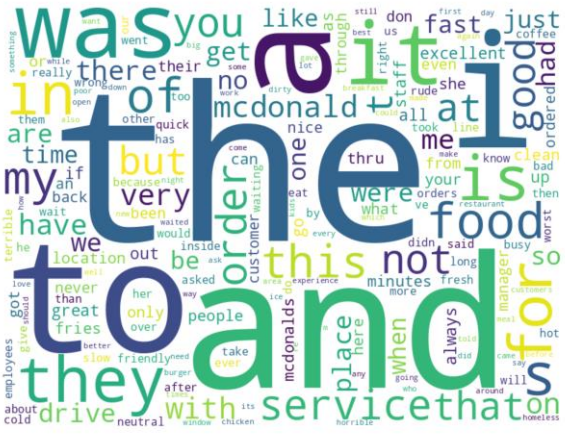

图 3.6 普通标签云

接着我使用 TF-IDF 方法，加上了词频特征，得到了图 3.7。从图中可以基本看出客户对麦当劳的整体评价偏向正面。

图 3.7 TF-IDF 标签云

下一步，我想进一步探究麦当劳的优缺点，我将用户评分进行二分类，即 0-2star 的评分为差评，4-5 的评分认为是好评。接着，我分别生成了新的标签云，如图 3.8，3.9 所示。可以看出，这些评价中名词和动词较多，而表示情感的形容词较少。尽管我们在好评中看到了 fast 的突出显示，这与麦当劳的快餐属性非常契合，然而从差评中我们很难总结出有效地结论以及用户对麦当劳的价值判断。

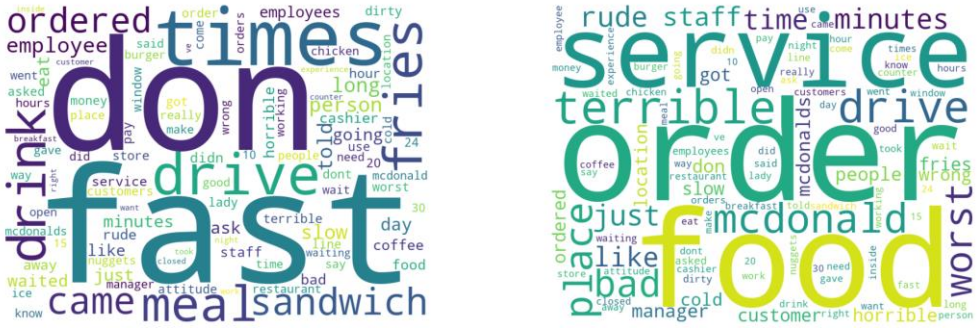

图 3.8-3.9 好评（左图）与差评（右图）标签云

于是进一步，我使用 ChatGPT4 从用户评论文本中分析生成了 600 个常用的形容词，然后只保留每条评论中的形容词，制作了新的好评与差评标签图，如图 3.10，3.11 所示。这一次，我得到了比较显著的结论。在好评方面，General，即一般，是用户对麦当劳最突出的评价，这也比较符合客观事实。除此之外，helpful，greasy，fresh，cheep 等服务型词汇较为常见，这说明麦当劳的周转模式和价格得到了一些用户的认可，healthy，delicious，等标识味道的词汇频率较小，这说

明大部分用户并不在意其味道。在差评方面，rude，bad，成为最突出的词汇，这说明麦当劳的服务不尽人意，而与印象相反的 slow 可能也说明了部分店铺在出餐效率上有待提升。

图 3.10-3.11 好评（左图）与差评（右图）标签云（600 形容词限定）

（2）交互设计

由于标签云更多的展示信息特征，因此本视图并没有设计用户交互选选项。

（3）视图分析

本视图通过循序渐进的方式展示了用户对于麦当劳的整体评价以及好评差评对麦当劳的单一评价，由于在第一部分已经详细阐述了对标签云的分析，此处不再赘述。

4 设计总结与心得

4.1 实验总结

在可视化实验中，我体验和尝试了整套可视化流程，从数据处理，到数据选择，再到确定可视化图表，设计用户交互，最后分析可视化图表总结出结论，我明白了可视化不只是调用 python 的库画图这么简单，用什么数据画图，画什么图，画图做什么，这些都是我在可视化实验中思考的问题，也帮助我提升了处理复杂数据场景的能力。

4.1.1 遇到的问题及处理

在实验中，我遇到了这样几个问题，并通过查阅资料和尝试妥善解决：

一是在使用 pyecharts 时，一开始无法正确渲染数据，后来发现这个库不能适配 dataframe 等数据格式，需要先转换成 list 格式才能正常渲染。

二是在设计平行坐标图时，店铺名一开始在左侧，可视化效果很差，几经尝试，在 toolboxs 里找到了对应的选项进行设置。

三是一开始我打算使用预训练模型对用户评价做二分类情感分析，但是调试训练后发现准确率仅仅 80%多，最后还是选择通过平均评分进行分类。

4.1.2 设计方案存在的不足

我认为本数据集的可视化方向还有可以完善的地方：

一是用户活跃度可视化，通过用户的评分总数，以及用户评价的时间曲线可以得到每家店铺的人流量信息以及用户活跃度信息。

二是评论时间的使用，可以可以通过用户评论时间的先后展现出店铺在不同时间内的服务质量，比如使用折线图展示趋势等。

4.2 实验心得

本次实验我选择的是麦当劳评论数据集，整体来说，这套数据集非常的原始，在可视化之前需要做大量的数据处理工作，而且本数据集的可视化思路需要深入思考才能得到，我也是一边实验一边思考，一步步深入，才能得到比较合理的可视化作品。不过，处理这样有挑战的数据集才能最大程度地发挥我的想象力，以至于我在完成实验后还能不断想到新点子。

与此同时，这次实验也让我认识到了箱线图，平行坐标图，雷达图等高维图

表，他们能够对更高维的数据进行可视化，同时我也认识到了视觉通道对可视化效果的重要作用。

4.3 意见与建议

我认为可视化实验难度适中，且总体工作量非常照顾学生，可选择性极强，让我既学到了可视化知识，又不会产生“坐牢”的体感，整体的实验体验非常出色，可谓出道即巅峰。

建议的话，或许可以尝试一下当下的可视化新技术，比如 Vizro 等库，或许可以让学生进一步理解不同可视化库之间的区别以及各自的优势，进一步拓宽知识领域。

原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

已阅读并同意以下内容。

判定为不合格的一些情形：

（1） 请人代做或冒名顶替者；

（2） 替人做且不听劝告者；

（3） 实验报告内容抄袭或雷同者；

（4） 实验报告内容与实际实验内容不一致者；

（5） 实验代码抄袭者。

作者签名：

附录一 实验代码

程序 1：Lab3.ipynb

```
from pyecharts.globals import ChartType
# the first graph to show the distribution of selected McDonald's Store
from pyecharts.charts import *
from pyecharts import options as opts
import pandas as pd
from pyecharts.globals import ThemeType

# Load the data
data = pd.read_csv('../dataset/McDonald/McDonald_s_Reviews.csv')

import reverse_geocoder as rg

def get_state_name(lat, lon):
    result = rg.search((lat, lon))

    if result:
        state_name = result[0]['admin1']
        return state_name
    return "Unknown"



# Example coordinates
lat = 38.456085
lon = -92.288368

state_name = get_state_name(lat, lon)
print(f"The coordinates (lat={lat}, lon={lon}) are in the state of {state_name}")

# Load the data
data = pd.read_csv('../dataset/McDonald/McDonald_s_Reviews.csv')
# Define a function to clean the 'rating' column and convert to numeric
def clean_rating(rating):
    if rating == "1 star":
```

```
        return float(rating.replace(' star', ''))
    else:
        return float(rating.replace(' stars', ''))

# Apply the function to the 'rating' column
data['rating'] = data['rating'].apply(clean_rating)

# if the store address is the same but the rating_count is different, we will change the highest
rating_count
# data = data.sort_values(by=['store_address', 'rating_count'], ascending=False)
# data = data.drop_duplicates(subset=['store_address'], keep='first')

for i in range(len(data)):
    # transform the 2,808    to 2808
    data.loc[i, 'rating_count'] = data.loc[i, 'rating_count'].replace(',', '')
    data.loc[i, 'rating_count'] = int(data.loc[i, 'rating_count'])
    if data.loc[i, 'rating_count'] == 1306:
        data.loc[i, 'rating_count'] = 1307
    if data.loc[i, 'rating_count'] == 1794:
        data.loc[i, 'rating_count'] = 1795
    if data.loc[i, 'rating_count'] == 998:
        data.loc[i, 'rating_count'] = 999
    if data.loc[i, 'rating_count'] == 1617:
        data.loc[i, 'rating_count'] = 1618
    if data.loc[i, 'rating_count'] == 3380:
        data.loc[i, 'rating_count'] = 3381
    if data.loc[i, 'rating_count'] == 2808:
        data.loc[i, 'rating_count'] = 2810
    if data.loc[i, 'rating_count'] == 5566:
        data.loc[i, 'rating_count'] = 5567
    if data.loc[i, 'rating_count'] == 1564:
        data.loc[i, 'rating_count'] = 1565
    if data.loc[i, 'rating_count'] == 19671:
        data.loc[i, 'rating_count'] = 19682
    if data.loc[i, 'rating_count'] == 5466:
```

```
            data.loc[i, 'rating_count'] = 5468

# Group by 'store_address' and calculate the mean rating
average_ratings = data.groupby(['store_address','latitude',
'longitude','rating_count'])['rating'].mean()



# add the state name to the data
average_ratings = average_ratings.reset_index()

average_ratings['state'] = average_ratings.apply(lambda row: get_state_name(row['latitude'],
row['longitude']), axis=1)
data_unique = average_ratings[['store_address', 'latitude', 'longitude','rating','state',"rating_count"]]

print(data_unique)

# Initialize the Parallel chart
parallel = Parallel(init_opts=opts.InitOpts(theme=ThemeType.LIGHT))

# Prepare the parallel data
parallel_data = [[i[0], i[4], i[5], i[3]] for i in data_unique.values]

# for all the parallel data[0], we only need the str before the first comma
for i in range(len(parallel_data)):
    parallel_data[i][0] = parallel_data[i][0].split(',')[0]

# Add schema (axes) to the parallel chart
parallel.add_schema(
    [
        opts.ParallelAxisOpts(dim=0,
                              name="store_address",
                              type_="category",),
        opts.ParallelAxisOpts(dim=1,
                              name="state",
                              type_="category",),
```

```
            {"dim": 2, "name": "rating count"},
            {"dim": 3, "name": "average rating"},


        ]
)

# Add the data to the parallel chart
parallel.add(
        "McDonald's Store",
        parallel_data,
        linestyle_opts=opts.LineStyleOpts(width=2, opacity=0.5),
)

# Add interactive elements
parallel.set_global_opts(
        title_opts=opts.TitleOpts(title="Parallel-Category"),
        toolbox_opts=opts.ToolboxOpts(),
        tooltip_opts=opts.TooltipOpts(trigger="item", formatter="{c}"),
        brush_opts=opts.BrushOpts(
            x_axis_index="all",
            brush_link="all",
            out_of_brush={"colorAlpha": 0.1},
            brush_type="lineX",
        ),
        datazoom_opts=[
            opts.DataZoomOpts(range_start=0, range_end=100, orient="vertical")
        ],
        visualmap_opts=opts.VisualMapOpts(
          max_= 4, min_= 2,is_piecewise=True,
        )
)

# Render the plot to a file
```

```
output_file = "../res/lab3/parallel_interactive.html"
parallel.render(output_file)

# The GEO graph to show the distribution of selected McDonald's Store

data_state = {}
data_state['California'] = 8
data_state['Nevada'] = 1
data_state['Utah'] = 1
data_state['Texas'] = 6
data_state['Illinois'] = 1
data_state['Florida'] = 7
data_state['Virginia'] = 1
data_state['Washington'] = 2
data_state['Pennsylvania'] = 1
data_state['New Jersey'] = 1
data_state['New York'] = 9


# Define state names and their coordinates (You need to provide this data)
states = {
    "Alabama": {"lat": 32.806671, "lon": -86.791130},
    "Alaska": {"lat": 61.370716, "lon": -152.404419},
    "Arizona": {"lat": 33.729759, "lon": -111.431221},
    "Arkansas": {"lat": 34.969704, "lon": -92.373123},
    "California": {"lat": 36.778261, "lon": -119.417932},
    "Colorado": {"lat": 39.550051, "lon": -105.782067},
    "Connecticut": {"lat": 41.597782, "lon": -72.755371},
    "Delaware": {"lat": 39.318523, "lon": -75.507141},
    "Florida": {"lat": 27.766279, "lon": -81.686783},
    "Georgia": {"lat": 33.040619, "lon": -83.643074},
    "Hawaii": {"lat": 21.094318, "lon": -157.498337},
    "Idaho": {"lat": 44.240459, "lon": -114.478828},
    "Illinois": {"lat": 40.349457, "lon": -88.986137},
    "Indiana": {"lat": 39.849426, "lon": -86.258278},
```

"Iowa": {"lat": 42.011539, "lon": -93.210526},
"Kansas": {"lat": 38.526600, "lon": -96.726486},
"Kentucky": {"lat": 37.668140, "lon": -84.670067},
"Louisiana": {"lat": 31.169546, "lon": -91.867805},
"Maine": {"lat": 44.693947, "lon": -69.381927},
"Maryland": {"lat": 39.063946, "lon": -76.802101},
"Massachusetts": {"lat": 42.230171, "lon": -71.530106},
"Michigan": {"lat": 43.326618, "lon": -84.536095},
"Minnesota": {"lat": 45.694454, "lon": -93.900192},
"Mississippi": {"lat": 32.741646, "lon": -89.678696},
"Missouri": {"lat": 38.456085, "lon": -92.288368},
"Montana": {"lat": 46.921925, "lon": -110.454353},
"Nebraska": {"lat": 41.125370, "lon": -98.268082},
"Nevada": {"lat": 38.313515, "lon": -117.055374},
"New Hampshire": {"lat": 43.452492, "lon": -71.563896},
"New Jersey": {"lat": 40.298904, "lon": -74.521011},
"New Mexico": {"lat": 34.840515, "lon": -106.248482},
"New York": {"lat": 42.165726, "lon": -74.948051},
"North Carolina": {"lat": 35.630066, "lon": -79.806419},
"North Dakota": {"lat": 47.528912, "lon": -99.784012},
"Ohio": {"lat": 40.388783, "lon": -82.764915},
"Oklahoma": {"lat": 35.565342, "lon": -96.928917},
"Oregon": {"lat": 44.572021, "lon": -122.070938},
"Pennsylvania": {"lat": 40.590752, "lon": -77.209755},
"Rhode Island": {"lat": 41.680893, "lon": -71.511780},
"South Carolina": {"lat": 33.856892, "lon": -80.945007},
"South Dakota": {"lat": 44.299782, "lon": -99.438828},
"Tennessee": {"lat": 35.747845, "lon": -86.692345},
"Texas": {"lat": 31.054487, "lon": -97.563461},
"Utah": {"lat": 40.150032, "lon": -111.862434},
"Vermont": {"lat": 44.045876, "lon": -72.710686},
"Virginia": {"lat": 37.769337, "lon": -78.169968},
"Washington": {"lat": 47.400902, "lon": -121.490494},
"West Virginia": {"lat": 38.491226, "lon": -80.954570},
"Wisconsin": {"lat": 44.268543, "lon": -89.616508},

```
    "Wyoming": {"lat": 42.755966, "lon": -107.302490}
}

geo_ = Geo(init_opts=opts.InitOpts(theme=ThemeType.LIGHT))
geo_.add_schema(maptype="美国")

# # Add coordinates for states
# for state, coord in states.items():
#     geo_.add_coordinate(name=state, longitude=coord["lon"], latitude=coord["lat"])
#     if data_state.get(state) != None:
#         geo_.add("State", [(state, data_state[state])], type_=ChartType.EFFECT_SCATTER)
#     else:
#         geo_.add("State", [(state, 0)], type_=ChartType.EFFECT_SCATTER)

# Add coordinates and heatmap data for stores
for i in range(len(data_unique)):
    # print(data_unique['store_address'][i], data_unique['rating_count'][i])
    print(data_unique['longitude'][i], data_unique['latitude'][i])
    geo_.add_coordinate(name=data_unique['store_address'][i],
longitude=data_unique['longitude'][i], latitude=data_unique['latitude'][i])
    geo_.add("McDonald Store", [(data_unique['store_address'][i], data_unique['rating'][i])],
type_=ChartType.EFFECT_SCATTER)

# Set labels and title
geo_.set_series_opts(
    label_opts=opts.LabelOpts(is_show=False),    # Show labels for states
)
geo_.set_global_opts(
    title_opts=opts.TitleOpts(title="McDonald's Locations in USA"),
    visualmap_opts=opts.VisualMapOpts(min_=2,max_=4,is_piecewise=True),
      toolbox_opts=opts.ToolboxOpts()

)

# Render the output file
```

```
geo_output_file = '../res/lab3/mcdonalds_locations_geo.html'
geo_.render(geo_output_file)

# Pie chart to show the average rating of selected McDonald's Store



# Prepare the data for the pie chart
rating_levels = ['Poor', 'Fair', 'Normal', 'Good', 'Excellent']
rating_counts = [len(data_unique[(data_unique['rating'] >= 2.0) & (data_unique['rating'] < 2.4)]),
                 len(data_unique[(data_unique['rating'] >= 2.4) & (data_unique['rating'] <
2.8)]),
                 len(data_unique[(data_unique['rating'] >= 2.8) & (data_unique['rating'] <
3.2)]),
                 len(data_unique[(data_unique['rating'] >= 3.2) & (data_unique['rating'] <
3.6)]),
                 len(data_unique[(data_unique['rating'] >= 3.6) & (data_unique['rating'] <=
4.0)])]

page = Page(layout=Page.SimplePageLayout)

# Create the rosetype pie chart
pie = (
    Pie(init_opts=opts.InitOpts(theme=ThemeType.LIGHT))
    .add(
        series_name="Average Rating Level",
        data_pair=[list(z) for z in zip(rating_levels, rating_counts)],
        radius=["20%", "75%"],
        center=["50%", "50%"],
        rosetype="area",
        label_opts=opts.LabelOpts(formatter="{b}: {c}"),
    )
    .set_global_opts(title_opts=opts.TitleOpts(title="Average Rating Level"),
                     toolbox_opts=opts.ToolboxOpts(),)
    .set_series_opts(label_opts=opts.LabelOpts(formatter="{b}: {c}"),)
)
```

```
# create a pie chart to show the number of stores in each state
state_counts = data_unique['state'].value_counts()
# divide the state_counts index and value to 2 lists
state_counts_index = state_counts.index.tolist()
state_counts_values = state_counts.values.tolist()
# Create the pie chart
pie_state = (
    Pie(init_opts=opts.InitOpts(theme=ThemeType.LIGHT))
    .add(
        series_name="Number of Stores by State",
        data_pair=[list(z) for z in zip(state_counts_index, state_counts_values)],
        radius=["20%", "75%"],
        center=["50%", "50%"],
        rosetype="area",
        label_opts=opts.LabelOpts(formatter="{b}: {c}"),
    )
    .set_global_opts(title_opts=opts.TitleOpts(title="Number of Stores by State"),
                     legend_opts=opts.LegendOpts(type_="scroll", pos_right="0%",
orient="vertical"),
                     toolbox_opts=opts.ToolboxOpts())
    .set_series_opts(label_opts=opts.LabelOpts(formatter="{b}: {c}"))
)

page.add(geo_)
page.add(parallel)
page.add(pie)
page.add(pie_state)


# Render the chart to an HTML file
pie.render("../res/lab3/pie.html")
pie_state.render("../res/lab3/pie_state.html")
page.render("../res/lab3/graph for McDonald's.html")
```

```
# Word Cloud 1 to show the most frequent words in the reviews
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from collections import Counter
import re

# Load the data from the CSV file
# Load the data
data = pd.read_csv('../dataset/McDonald/McDonald_s_Reviews.csv')

# Function to clean and split text into words
def clean_and_split(text):
    if not isinstance(text, str):
        return []
    # Remove non-alphabetic characters and split into words
    words = re.findall(r'\b[a-zA-Z]+\b', text.lower())
    return words
# Define a function to clean the 'rating' column and convert to numeric
def clean_rating(rating):
    if rating == "1 star":
        return float(rating.replace(' star', ''))
    else:
        return float(rating.replace(' stars', ''))

# Apply the function to the 'rating' column
data['rating'] = data['rating'].apply(clean_rating)



# Extract the 'review' column
reviews = data['review']

# regard rating > 3 as positive, rating < 3 as negative
reviews_positive = reviews[data['rating'] > 3]
```

```
reviews_negative = reviews[data['rating'] < 2]

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=100)

# Apply TF-IDF to the reviews
tfidf_matrix = vectorizer.fit_transform(reviews.fillna(""))    # Filling missing values with empty strings
tfidf_matrix_positive = vectorizer.fit_transform(reviews_positive.fillna(""))    # Filling missing values with empty strings
tfidf_matrix_negative = vectorizer.fit_transform(reviews_negative.fillna(""))    # Filling missing values with empty strings

# Summarize the TF-IDF scores for each word
word_scores = tfidf_matrix.sum(axis=0)
words = vectorizer.get_feature_names_out()
word_freq_tfidf = {words[i]: word_scores[0, i] for i in range(len(words))}

# Summarize the TF-IDF scores for each positive word
word_scores_positive = tfidf_matrix_positive.sum(axis=0)
words_positive = vectorizer.get_feature_names_out()
word_freq_tfidf_positive = {words_positive[i]: word_scores_positive[0, i] for i in range(len(words_positive))}
# Summarize the TF-IDF scores for each negative word
word_scores_negative = tfidf_matrix_negative.sum(axis=0)
words_negative = vectorizer.get_feature_names_out()
word_freq_tfidf_negative = {words_negative[i]: word_scores_negative[0, i] for i in range(len(words_negative))}

# Generate word cloud using the TF-IDF scores
wordcloud_tfidf = WordCloud(width=800, height=600, background_color='white').generate_from_frequencies(word_freq_tfidf)
wordcloud_tfidf_positive = WordCloud(width=800, height=600, background_color='white').generate_from_frequencies(word_freq_tfidf_positive)
wordcloud_tfidf_negative = WordCloud(width=800, height=600,
```

```
background_color='white').generate_from_frequencies(word_freq_tfidf_negative)

# Plotting the Word Cloud
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud_tfidf, interpolation='bilinear')
plt.axis('off')

# Save the word cloud image
wordcloud_image_path_tfidf = '../res/wordcloud_tfidf.png'
plt.savefig(wordcloud_image_path_tfidf)

# Plotting the Word Cloud positive
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud_tfidf_positive, interpolation='bilinear')
plt.axis('off')
# save the word cloud image
wordcloud_image_path_tfidf_positive = '../res/wordcloud_tfidf_positive.png'
plt.savefig(wordcloud_image_path_tfidf_positive)

# Plotting the Word Cloud negative
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud_tfidf_negative, interpolation='bilinear')
plt.axis('off')

# Save the word cloud image
wordcloud_image_path_tfidf_negative = '../res/wordcloud_tfidf_negative.png'
plt.savefig(wordcloud_image_path_tfidf_negative)

# Word Cloud 2 in 300 adj words
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

```
# Load the data
data = pd.read_csv('../dataset/McDonald/McDonald_s_Reviews.csv')

# Define a function to clean the 'rating' column and convert to numeric
def clean_rating(rating):
    if "1 star" in rating:
        return 1
    else:
        return float(rating.replace(' stars', ''))

# Apply the function to the 'rating' column
data['rating'] = data['rating'].apply(clean_rating)

# Filter reviews based on rating
reviews_positive = data[data['rating'] > 3]['review'].fillna("")
reviews_negative = data[data['rating'] < 3]['review'].fillna("")

# List of common adjectives in restaurant reviews
common_adjectives = [
    "good", "great", "bad", "delicious", "tasty", "poor", "excellent", "terrible", "nice", "friendly",
    "dirty", "clean", "slow", "fast", "fresh", "old", "hot", "cold", "spicy", "sweet", "savory",
"rich",
    "creamy", "crispy", "juicy", "sour", "bitter", "yummy", "flavorful", "tangy", "zesty",
"smoky",
    "grilled", "baked", "roasted", "fried", "steamed", "raw", "crunchy", "soft", "hard", "dense",
    "light", "hearty", "healthy", "unhealthy", "salty", "bland", "seasoned", "spiced", "zippy",
"buttery",
    "cheesy", "chocolaty", "nutty", "fruity", "meaty", "vegetarian", "vegan", "gluten-free",
"organic",
    "local", "exotic", "traditional", "innovative", "creative", "unique", "classic", "modern",
"fancy",
    "casual", "cozy", "comfortable", "elegant", "stylish", "busy", "quiet", "loud", "crowded",
"empty",
    "spacious", "compact", "intimate", "romantic", "family-friendly", "welcoming", "unfriendly",
"rude",
```

"polite", "attentive", "neglectful", "helpful", "unhelpful", "knowledgeable", "inexperienced",
"professional", "amateurish", "quick", "lengthy", "timely", "late", "early", "expensive", "cheap",
"affordable", "overpriced", "worthwhile", "valuable", "pricey", "economical", "luxurious", "opulent",
"lavish", "simple", "plain", "ornate", "decorative", "colorful", "drab", "bright", "dark", "warm",
"cool", "airy", "stuffy", "smoky", "fragrant", "aromatic", "odorous", "scented", "musty", "fresh",
"stale", "moist", "dry", "humid", "damp", "soggy", "wet", "arid", "crispy", "chewy", "tender",
"tough", "rubbery", "succulent", "lean", "fatty", "greasy", "oily", "smooth", "gritty", "fine",
"coarse", "thick", "thin", "hefty", "lightweight", "substantial", "insubstantial", "filling",
"satisfying", "unsatisfying", "appetizing", "unappetizing", "delectable", "disgusting", "enjoyable",
"unenjoyable", "pleasant", "unpleasant", "tantalizing", "uninviting", "inviting", "appealing",
"unappealing", "enticing", "repulsive", "luscious", "revolting", "enticing", "alluring", "off-putting",
"gorgeous", "unattractive", "stunning", "dreary", "vibrant", "dull", "radiant", "gloomy", "shiny",
"matte", "glossy", "polished", "unpolished", "sleek", "rough", "ragged", "neat", "messy", "orderly",
"chaotic", "organized", "disorganized", "compact", "spacious", "cramped", "roomy", "airy", "claustrophobic",
"open", "closed", "expansive", "narrow", "broad", "wide", "constricted", "expansive", "capacious",
"generous", "stingy", "ample", "limited", "abundant", "scarce", "plentiful", "meager", "adequate",
"insufficient", "sufficient", "lacking", "complete", "incomplete", "whole", "partial", "comprehensive",
"limited", "detailed", "sketchy", "specific", "general", "particular", "vague", "precise", "inaccurate",
"accurate", "true", "false", "real", "fake", "authentic", "imitation", "genuine", "artificial",
"natural", "synthetic", "manmade", "handmade", "machine-made", "crafted", "manufactured", "produced",
"created", "designed", "planned", "improvised", "intentional", "accidental", "deliberate",

```
"spontaneous",
    "calculated", "unplanned", "organized", "disordered", "systematic", "haphazard",
"methodical", "random",
    "consistent", "inconsistent", "uniform", "varied", "similar", "different", "alike", "unlike",
    "comparable", "incomparable", "equivalent", "unequal", "equal", "superior", "inferior",
"better",
    "worse", "improved", "degraded", "enhanced", "diminished", "upgraded", "downgraded",
"advanced",
    "primitive", "modern", "antique", "contemporary", "historic", "new", "old", "fresh", "stale",
    "recent", "ancient", "current", "outdated", "latest", "earliest", "first", "last", "initial",
    "final", "beginning", "end", "start", "finish", "opening", "closing", "launch", "conclusion",
    "introduction", "completion", "inception", "termination", "genesis", "demise", "birth",
"death",
    "origin", "conclusion", "foundation", "destruction", "creation", "annihilation", "formation",
"dissolution",
    "assembly", "disassembly", "construction", "deconstruction", "building", "demolition",
"erecting", "razing",
    "raising", "lowering", "elevating", "depressing", "ascending", "descending", "climbing",
    "descending", "climbing", "falling", "rising", "dropping", "plummeting", "soaring",
"plunging", "skyrocketing",]

# Function to filter text for these adjectives
def filter_adjectives(text):
    words = re.findall(r'\b[a-zA-Z]+\b', text.lower())
    return ' '.join(word for word in words if word in common_adjectives)

# Apply adjective filtering to the reviews
reviews_positive_adj = reviews_positive.apply(filter_adjectives)
reviews_negative_adj = reviews_negative.apply(filter_adjectives)

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=100)

# Apply TF-IDF to the adjective reviews
tfidf_matrix_positive = vectorizer.fit_transform(reviews_positive_adj)
```

```
tfidf_matrix_negative = vectorizer.fit_transform(reviews_negative_adj)

# Summarize the TF-IDF scores for positive and negative adjectives
word_scores_positive = tfidf_matrix_positive.sum(axis=0)
word_scores_negative = tfidf_matrix_negative.sum(axis=0)

words_positive = vectorizer.get_feature_names_out()
words_negative = vectorizer.get_feature_names_out()

word_freq_tfidf_positive = {words_positive[i]: word_scores_positive[0, i] for i in
range(len(words_positive))}
word_freq_tfidf_negative = {words_negative[i]: word_scores_negative[0, i] for i in
range(len(words_negative))}

# Generate word clouds
wordcloud_tfidf_positive = WordCloud(width=800, height=600,
background_color='white').generate_from_frequencies(word_freq_tfidf_positive)
wordcloud_tfidf_negative = WordCloud(width=800, height=600,
background_color='white').generate_from_frequencies(word_freq_tfidf_negative)

# Plotting the Word Clouds
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud_tfidf_positive, interpolation='bilinear')
plt.axis('off')
plt.savefig('../res/wordcloud_tfidf_positive.png')

plt.figure(figsize=(10, 8))
plt.imshow(wordcloud_tfidf_negative, interpolation='bilinear')
plt.axis('off')
plt.savefig('../res/wordcloud_tfidf_negative.png')

import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.metrics import accuracy_score, classification_report

# Load the data
data = pd.read_csv('../dataset/McDonald/McDonald_s_Reviews.csv')

# Define a function to clean the 'rating' column and convert to numeric
def clean_rating(rating):
    if "1 star" in rating:
        return 1
    elif "2 stars" in rating:
        return 2
    elif "3 stars" in rating:
        return 3
    elif "4 stars" in rating:
        return 4
    elif "5 stars" in rating:
        return 5
    else:
        return 0    # For any non-standard ratings

# Apply the function to the 'rating' column
data['numeric_rating'] = data['rating'].apply(clean_rating)

# Consider ratings of 4 or 5 stars as positive (1), and the rest as negative (0)
data['sentiment'] = data['numeric_rating'].apply(lambda x: 1 if x >= 4 else 0)

# Split the data into features and target variable
X = data['review'].fillna(" ")    # Filling missing reviews with a blank space
y = data['sentiment']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the CountVectorizer
vectorizer = CountVectorizer(stop_words='english')
```

```
# Fit and transform the training data
X_train_vectorized = vectorizer.fit_transform(X_train)

# Transform the test data
X_test_vectorized = vectorizer.transform(X_test)

# Initialize the Multinomial Naive Bayes classifier
clf = MultinomialNB()

# Train the classifier
clf.fit(X_train_vectorized, y_train)

# Predict on the test set
y_pred = clf.predict(X_test_vectorized)

# Calculate accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Classification Report:\n", report)

# save the model
import pickle
pickle.dump(clf, open("../res/lab3/mcdonalds_sentiment_model.pkl", 'wb'))



# load the model to predict
model = pickle.load(open("../res/lab3/mcdonalds_sentiment_model.pkl", 'rb'))
```