HW9-作业讲评

王瑞环

1.1 均值与标准差

- 使用groupby和agg

```
display(df_stustress.groupby(
    'How would you rate your stress levels?'
).agg(['mean', 'std']))
```

1.2 计数与比值计算

- 用&连接条件并计数

```
load_str = 'how would you rate your study load?'
stress_str = 'How would you rate your stress levels?'

count_high_load = df_stustress[
    (df_stustress[load_str] >= 4) & (df_stustress[load_str] <= 5)
].groupby(stress_str).size()
count_low_load = df_stustress[
    (df_stustress[load_str] >= 1) & (df_stustress[load_str] <= 2)
].groupby(stress_str).size()
result_df = pd.DataFrame({
    'High Load Count': count_high_load,
    'Low Load Count': count_low_load
})
result_df['Load Count Ratio'] = \
    result_df['High Load Count'] / result_df['Low Load Count']
result_df.index.name = 'Stress Level'
display(result_df)
```

1.2 计数与比值计算

- 用cut切分

```
# 汤齐宇
df_stustress['study_load_category'] = pd.cut(
    df_stustress['how would you rate your study load?'],
    bins=[0, 2, 3, 5], labels=['Low', 'median', 'High'])

# 按照压力等级和学习负担分类进行分组，并计算人数
grouped_counts = df_stustress.groupby([
    'How would you rate your stress levels?',
    'study_load_category'
]).size().unstack(fill_value=0)

# 计算比值
grouped_counts['Ratio'] \
    = grouped_counts['High'] / grouped_counts['Low']
grouped_counts[['High', 'Low', 'Ratio']]
```

1.2 计数与比值计算

- 用isin作为条件并计数

```
# 卢浩楠
high_load = [4, 5]
low_load = [1, 2]
study_load_df = pd.DataFrame(
    index=df.index.unique(),
    columns=['High Load Count', 'Low Load Count'])
study_load_df.index.name = 'Stress Level'
study_load_df['High Load Count'] = \
    df[df['how would you rate your study load?'].isin(
        high_load)].groupby('How would you rate your stress levels?').size()
study_load_df['Low Load Count'] = \
    df[df['how would you rate your study load?'].isin(
        low_load)].groupby('How would you rate your stress levels?').size()

study_load_df['Load Count Ratio'] = \
    study_load_df['High Load Count'] / study_load_df['Low Load Count']

study_load_df.sort_index()
```

1.3 散点图

- 多重索引groupby，使用get_level_values获取索引值

```
df_stustress_2 = df_stustress.groupby([
    'How would you rate your stress levels?',
    'Kindly Rate your Sleep Quality 😴']).size()
stress_levels = df_stustress_2.index.get_level_values(0)
sleep_quality = df_stustress_2.index.get_level_values(1)
counts = df_stustress_2.values

plt.scatter(stress_levels, sleep_quality, s=counts*80)
plt.xlabel('Stress Levels')
plt.ylabel('Sleep Quality')
plt.show()
```

1.3 散点图

- 多重索引groupby，使用unstack转换为二维表

```
# 汤齐宇
df = df_stustress.groupby([
    'How would you rate your stress levels?',
    'Kindly Rate your Sleep Quality 😴'])[
        'How many times a week do you suffer headaches 😕?'
    ].count().unstack(fill_value=0)
plt.figure(figsize=(9,7))
plt.scatter(
    [i for i in df.index for j in df.columns],
    [j for i in df.index for j in df.columns],
    s=df*200)
plt.ylabel('Sleep Quality')
plt.xlabel('Stress level')
plt.show()
```

1.3 散点图

- 使用meshgrid取索引

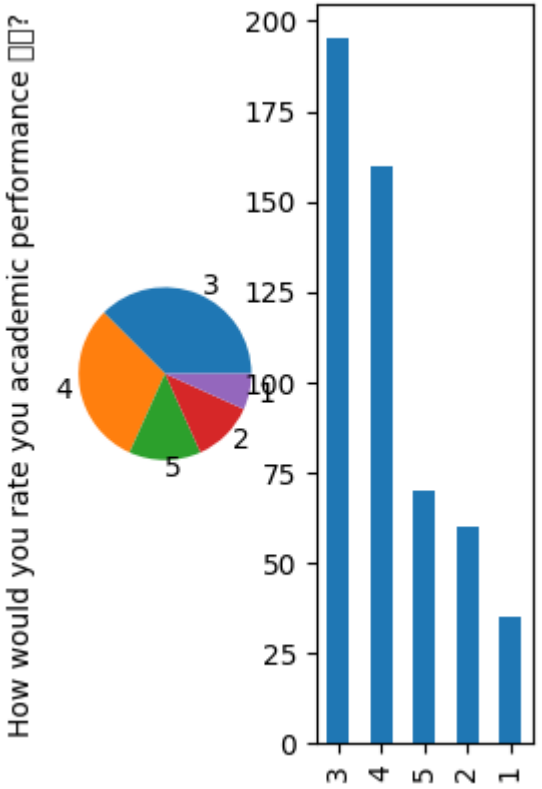
```
# 程一哲
grouped2 = df_stustress.groupby(
    by=[columns[0]]
    )[columns[5]].value_counts().unstack().fillna(0)

X, Y = np.meshgrid(grouped.columns, grouped.index)
display(X, Y)
plt.scatter(X, Y, s=grouped2.values*80)
```

```
array([[1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5]], dtype=int64)

array([[1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4],
       [5, 5, 5, 5, 5]], dtype=int64)
```

1.4 分布情况可视化

- "性价比"最高的写法

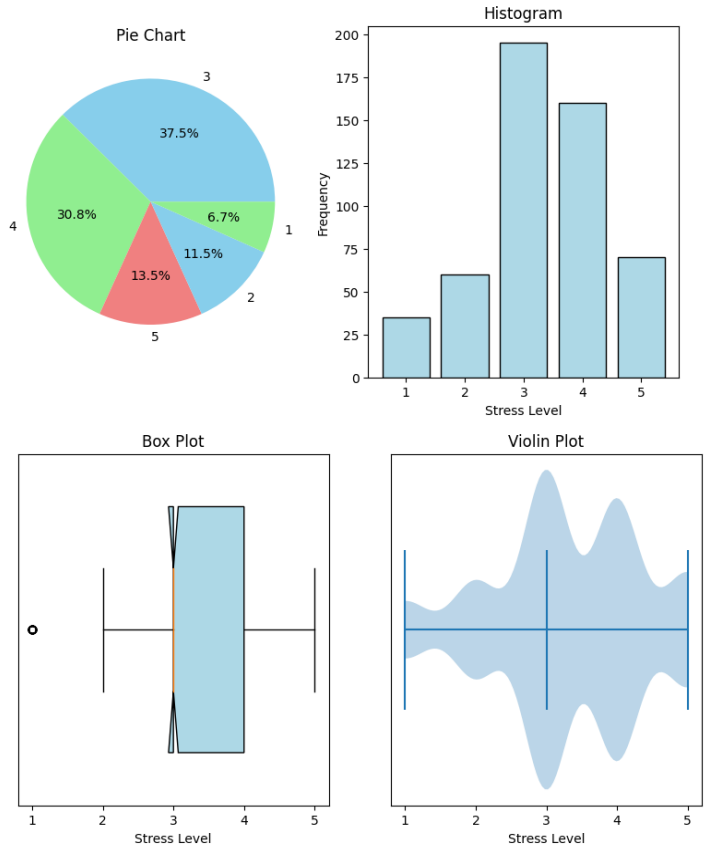
```
data_plot = df_stustress[
    'How would you rate you academic performance 👩‍🎓?'
].value_counts()

plt.subplot(141)
data_plot.plot.pie()
plt.subplot(142)
data_plot.plot.bar()
plt.show()
```



1.4 分布情况可视化

- 参考输出的写法

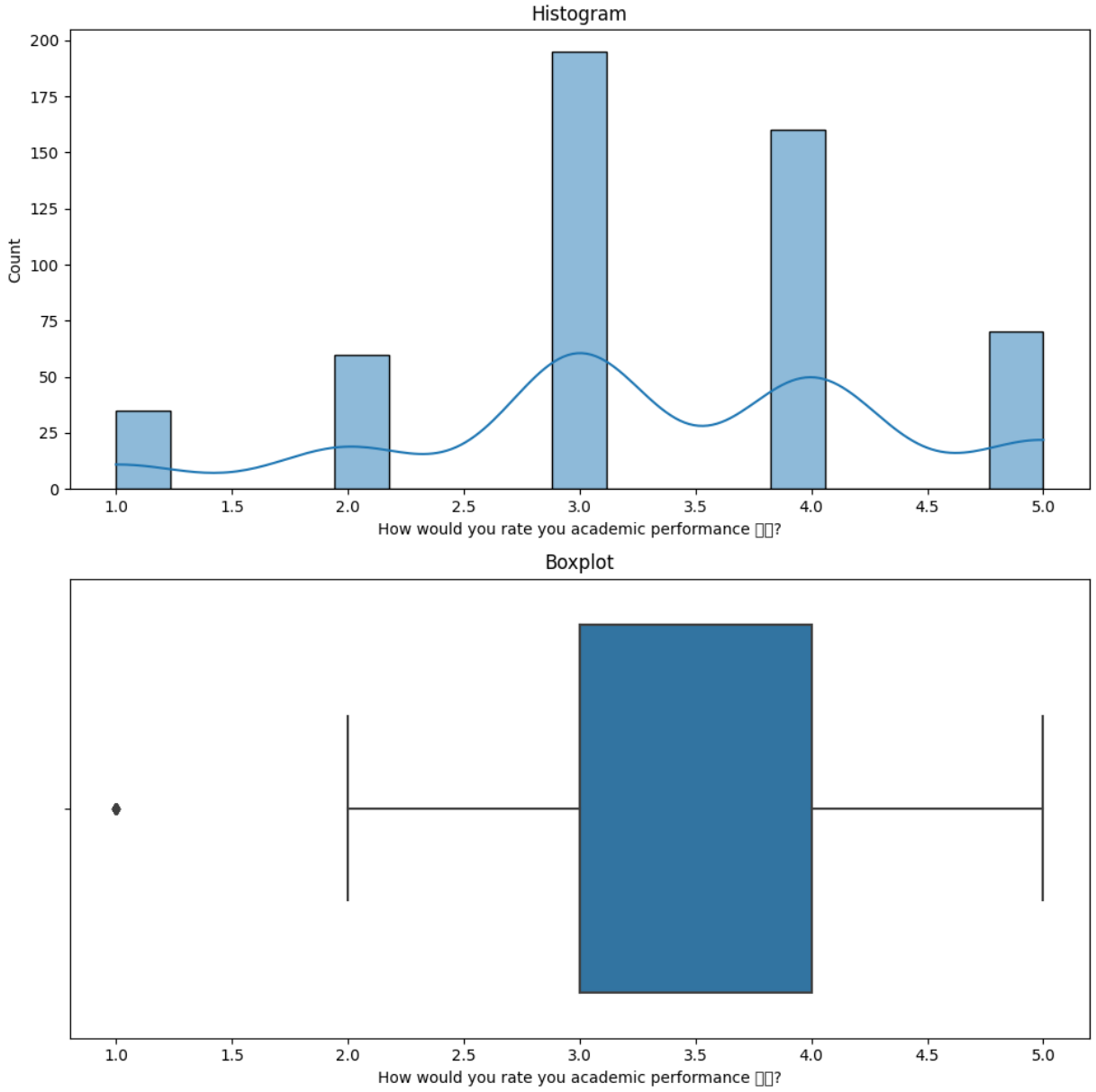
```
data_plot = df_stustress['How would you
rate you academic performance 👩‍🏫?']

plt.figure(figsize=(20, 5))

plt.subplot(141)
plt.pie(
    x=data_plot.value_counts(),
    labels=data_plot.value_counts().index,
    autopct='%1.1f%%',
    colors=['skyblue', 'lightgreen',
'lightcoral'])
plt.title('Pie Chart')

plt.subplot(142)
plt.hist(
    x=data_plot,
    bins=range(1, 7),
    align='left',
    rwidth=0.8,
    color='lightblue',
    edgecolor='black')
plt.title('Histogram')
plt.xlabel('Stress Level')
plt.ylabel('Frequency')

plt.subplot(143)
plt.boxplot(
    x=data_plot,
    patch_artist=True,
    notch=True,
    vert=False,
    widths=0.7,
    boxprops=dict(facecolor='lightblue'))
plt.title('Box Plot')
plt.yticks([])
plt.xlabel('Stress Level')

plt.subplot(144)
plt.violinplot(
    dataset=data_plot,
    vert=False,
    widths=0.7,
    showmedians=True)
plt.title('Violin Plot')
plt.yticks([])
plt.xlabel('Stress Level')

plt.show()
```



1.4 分布情况可视化

- 使用seaborn

```
# 宋林烜
data_plot = df_stustress['How would you rate
you academic performance 👨‍🦱?']
import seaborn as sns
fig, axs = plt.subplots(2, 1, figsize=(10, 10))
sns.histplot(data_plot, kde=True, ax=axs[0])
axs[0].set_title('Histogram')
sns.boxplot(x=data_plot, ax=axs[1])
axs[1].set_title('Boxplot')
plt.tight_layout()
plt.show()
```

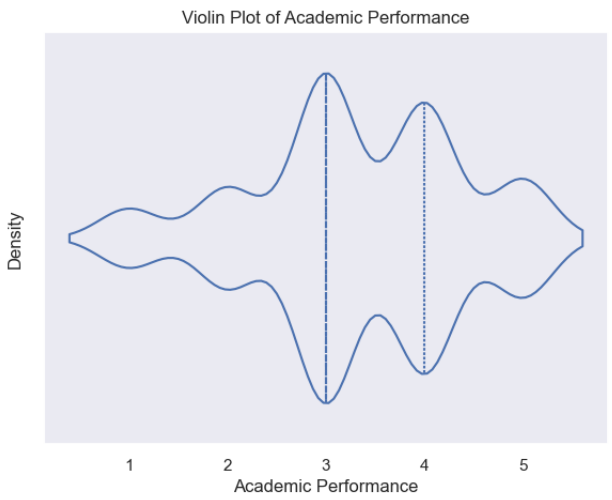


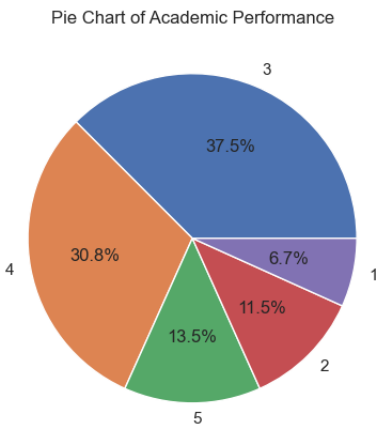
1.4 分布情况可视化

- 使用seaborn

```
# 马佳昊
import seaborn as sns
sns.set_theme(style="dark")
plt.figure(figsize=(15, 5))
target_df = pd.DataFrame(
    df_stustress['How would you rate you academic performance 👩?'].value_counts())
target_df['level'] = target_df.index
plt.subplot(1, 2, 1)
p = sns.violinplot(
    data=df_stustress,
    x = 'How would you rate you academic performance 👩?',
    split=False, inner="quart", fill=False)
p.set_xlabel('Academic Performance')
p.set_ylabel('Density')
p.set_title('Violin Plot of Academic Performance')
```

```
plt.subplot(1, 2, 2)
plt.pie(
    df_stustress['How would you rate you academic performance 👩?'].value_counts(),
    autopct='%1.1f%%',
    labels = df_stustress['How would you rate you academic performance 👩?'].value_counts().index.to_list())
plt.title('Pie Chart of Academic Performance')
plt.show()
```





2.1.1 表合并与空值处理

```
df = pd.merge(content_df, category_df, on='NewsID')
df = df.dropna()
```

2.1.2 文本预处理

- 正则表达式处理标点

```
from stopwords import ENGLISH_STOP_WORDS
def preprocess(x: str) -> str:
    x = x.lower()

    puncs = re.compile(r'[\'"\\\|\{\}\[\]:;,./<>\?`~!@#\$%\^&\*()_\+-=]')
    x = puncs.sub(' ', x)

    word_list = x.split()
    word_list = [word for word in word_list if word not in ENGLISH_STOP_WORDS]
    x = ' '.join(word_list)
    # x = re.sub(r'\b\w{1,3}\b', ' ', x)
    # x = re.sub(r'\s+', ' ', x)

    return x
```

2.1.2 文本预处理

- 使用str.maketrans和string标准库自带的标点符号集

```python
# 卢浩楠
import string
from stopwords import ENGLISH_STOP_WORDS

def preprocess(x: str) -> str:
    x = x.lower()
    x = x.translate(str.maketrans('', '', string.punctuation))  # Remove punctuation
    # Split the text into words and remove stopwords
    words = x.split()
    words = [word for word in words if word not in ENGLISH_STOP_WORDS]
    # Join the words back into a single string
    x = ' '.join(words)

    return x
```

2.1.2 文本预处理

- 标准化与词干提取

```
# 王子宸
normalization_rules = {
    "colour": "color",
    "favourite": "favorite",
    "organise": "organize",
    "realise": "realize",
    "recognise": "recognize",
    "analyse": "analyze",
    "behaviour": "behavior",
    # more ...
}

def normalize(word: str) -> str:
    return normalization_rules.get(word, word)
```

```
def stemize(word: str) -> str:
    special_cases = {
        'ies': 'y',
        'ing': '',
        'ed': '',
    }
    suffixes = ['ly', 'ious', 'ive', 'es', 's', 'ment']

    for special_suffix, replacement in special_cases.items():
        if word.endswith(special_suffix):
            base = word[:-len(special_suffix)] + replacement
            if len(base) > 3:
                return base
            else:
                return word

    for suffix in suffixes:
        if word.endswith(suffix):
            return word[:-len(suffix)]

    return word
```

2.2 TF-IDF

- 使用Counter，运行速度~0.5s

```
from collections import Counter
def getTFIDF(corpus: List[str]) -> Tuple[np.ndarray, Dict[str, int]]:
    # Get word2idx ...
    df_counter = Counter()

    for sentence in corpus:
        df_counter.update(set(sentence.split()))
    for i, sentence in enumerate(corpus):
        lens = len(sentence.split())
        word_count = Counter(sentence.split())
        for word in sentence.split():
            tf_word = word_count[word] / lens
            idf_word = math.log((num_sentence) / (df_counter[word]))
            tfidf[word2idx[word], i] = tf_word * idf_word
    return tfidf, word2idx
```

2.2 TF-IDF

- 使用set和defaultdict，运行速度~0.5s

```
# 王子宸
from collections import defaultdict
def getTFIDF(corpus: List[str]) -> Tuple[np.ndarray, Dict[str, int]]:
    # Get word2idx ...
    d = defaultdict(int)
    for sentence in corpus:
        for word in set(sentence.split()):
            d[word] += 1
    for i, sentence in enumerate(corpus):
        words = sentence.split()
        tf_dot_j = len(words)

        words_count = defaultdict(int)
        for word in words:
            words_count[word] += 1
        for word, count in words_count.items():
            tf = count / tf_dot_j
            idf = math.log(num_sentence / d[word])
            tfidf[word2idx[word], i] = tf * idf
    return tfidf, word2idx
```

2.2 TF-IDF

- 使用numpy广播运算（需处理NaN），运行速度~1.5s

```
# 曾为帅
def getTFIDF(corpus: List[str]) -> Tuple[np.ndarray, Dict[str, int]]:
    # Get word2idx ...
    for i in range(num_sentence):
        for word in corpus[i].split():
            tfidf[word2idx[word]][i]+=1
    tf=tfidf/tfidf.sum(axis=0)
    tf[np.isnan(tf)]=0
    idf=np.log(num_sentence/np.count_nonzero(tfidf,axis=1))[:,np.newaxis]
    tfidf=tf*idf

    return tfidf, word2idx
```

2.2 TF-IDF

- 使用numpy广播运算（需处理NaN），运行速度~3s

```
# 邓欣宇
def getTFIDF(corpus: List[str]) -> Tuple[np.ndarray, Dict[str, int]]:
    # Get word2idx ...
    tf = np.zeros((num_words,num_sentence))
    idf = np.zeros((num_words,num_sentence))
    # tf
    for j, sentence in enumerate(corpus):
        words = sentence.split()
        for word in words:
            tf[word2idx[word]][j] += 1
            idf[word2idx[word]][j] = 1
    tf = tf / tf.sum(axis=0)
    np.nan_to_num(tf,0) #避免除0导致的nan
    idf = np.log(num_sentence / idf.sum(axis=1))
    tfidf = tf*idf[:,np.newaxis]
    return tfidf, word2idx
```

2.2 TF-IDF

- 使用DataFrame操作，运行速度~3s

```
# 姬钰
def getTFIDF(corpus: List[str]) -> Tuple[np.ndarray, Dict[str, int]]:

    contents=pd.DataFrame(list(corpus),columns=["content"])
    splited=contents["content"].str.split(
            " ",expand=True
        ).stack().reset_index().groupby(
            by=['level_0',0]
        ).count().unstack(fill_value=0)
    idf=-1*np.log((splited!=0).sum().divide(len(splited)))
    idf.index=idf.index.droplevel(None)
    count=splited.copy()
    count.columns=count.columns.droplevel(None)
    tf=count.divide(count.sum(axis=1),axis=0)
    tfIdf=tf.multiply(idf,axis=1)
    columns = tfIdf.columns.tolist()
    word2idx={column: index for index, column in enumerate(columns)}
    tfidf=tfIdf.values.T
    return tfidf, word2idx
```

2.3 KMeans

- 以下代码的运行时间
  - 低版本sklearn ~40s
  - 高版本sklearn ~4s

```
km_cluster = KMeans(n_clusters=n_clusters, init='k-means++', max_iter=100)
km_cluster.fit(X_normalized)
```

- 原因：Kmeans参数n_init默认为10，高版本默认为'auto'，参数含义为以不同的起始中心点开始运行Kmeans算法的次数

2.4 关键词提取

- 设M为单词数，则每个聚类中心表示为一个长为M的向量
- 向量在第i维上的坐标即为单词i在该主题下的权重

```
def get_frequencies(km: KMeans, cluster_index: int) -> Dict[str, float]:
    frequencies = dict()

    term_frequencies = km.cluster_centers_[cluster_index]
    frequencies = {idx2word[i]: term_frequencies[i] for i in range(len(word2idx))}

    return frequencies
```

3.1 LSA

- make_pipeline：获取一个Pipeline类型的变量，顺序在数据上执行所有组成部分
- 与之类似的功能模块：
  - torch.nn.Sequential
  - torchvision.transforms.Compose

```
lsa = make_pipeline(TruncatedSVD(n_components=5), Normalizer(copy=False))
t0 = time.time()

X_lsa = lsa.fit_transform(X_normalized)
km_lsa = KMeans(n_clusters=5, init='k-means++', max_iter=100)
km_lsa.fit(X_lsa)

print("done in %0.3fs" % (time.time() - t0))
evaluate_kmeans(km_lsa)
```

3.1 LSA

- Normalizer不可以求逆变换，但归一化对数据相对大小比较并不会产生影响
- TruncatedSVD：(N, M) --> (N, K)
- inverse_transform：(N', K) --> (N', M)

```
def get_frequencies_lsa(km: KMeans, cluster_index: int) -> Dict[str, float]:
    frequencies = dict()
    term_frequencies = lsa.steps[0][1].inverse_transform(
        km.cluster_centers_)[cluster_index]
    frequencies = {idx2word[i]: term_frequencies[i] for i in range(len(word2idx))}
    return frequencies
```

3.2.1 矢量场流线图

```
x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)

Ex1, Ey1 = electric_field(q1, r0_1, X, Y)
Ex2, Ey2 = electric_field(q2, r0_2, X, Y)
Ex = Ex1 + Ex2
Ey = Ey1 + Ey2
```

```
plt.figure(figsize=(5, 5))
plt.scatter(r0_1[0], r0_1[1], s=100,
            color='red', marker='o',
            label='Positive Charge',
            zorder=2)
plt.scatter(r0_2[0], r0_2[1], s=100,
            color='yellow', marker='o',
            label='Negative Charge',
            zorder=2)
plt.streamplot(X, Y, Ex, Ey, density=0.7,
               linewidth=1, zorder=1,
               # broken_streamlines=False
               # 用于保证流线连续性
               # 低版本matplotlib不支持
)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Electric Field Lines')
plt.legend()
plt.show()
```

3.2.2 三维曲面

```
u, v = np.linspace(0, 2*np.pi, 40), np.linspace(0, 2*np.pi, 40)
ux, vx =  np.meshgrid(u,v)
x, y, z = surf(ux, vx)

fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111, projection='3d')
plot = ax.plot_surface(x, y, z, antialiased=True)
plt.show()
```