Contents

1 Introduction

With the recent renewal of the MBTI's popularity, posts or analyses about it have become widespread on public platforms. It seems that MBTI has become a way people use at first meetings.

The Myers-Briggs Type Indicator (MBTI) is a psychological framework primarily used to describe and measure personality traits by categorizing individuals into four dichotomous dimensions. In essence, it aims to reflect how people perceive the world and make decisions based on their internal experiences(Yang, 2022).

To determine one's type, the test asks a series of questions, usually focused on a person's tendencies or preferences. Based on the responses, individuals receive a result represented by four letters, each corresponding to one of the four bipolar dimensions. According to Pittenger (2005), these four dichotomous dimensions classify individuals as either extraverted (E) or introverted (I), sensing (S) or intuitive (N), thinking (T) or feeling (F), and judging (J) or perceiving (P). Combinations of the four preferences generate one of 16 personality types (e.g., ESFJ, ENFP, INTP, ISFJ), each associated with distinct behavioral tendencies, reflecting differences in attitudes, orientation, and decision-making styles. Also, on the official website, it will provide the future career options for you, like INFP, which may be more suitable for being an author. The percentage of each standard will also show in the final result.

And based on the 16 personality types, it can be divided into four groups . As each role depends on two shared traits that they have in common, the four groups are: Analysts, who share the Intuitive and Thinking traits (NT); Diplomats, who share the Intuitive and Feeling traits (NF); Sentinels, who share the Observant and Judging traits (S_J); and Explorers, who share the Observant and Prospecting traits (S_P) (16Personalities, n.d.). And the Myers-Briggs Type Indicator (MBTI) sorts them into four colors, as the purple color represents Analysts, the green color represents Diplomats, the blue color represents Sentinels, and the yellow color represents Explorers.

The MBTI provides a widely recognized and accessible way to understand personality, making it a useful foundation for further behavioral and data-driven analysis. However, it has also attracted criticism as some thought the result of the MBTI test was not reliable. As it may bring some stereotypes when forming impressions of others.

Given its popularity and sustaining controversy, our group decided to explore the MBTI more deeply and explore the rationality of this test.

2 Objective

1. Evaluate the reliability of the MBTI from a statistical perspective.
2. Explore the potential application of MBTI in social media behavior analysis.
3. Help people better understand personality traits and behavioral patterns.
4. Helping people eliminate stereotypes caused by MBTI personality types.

3 Problem Statements

1. Are the results of the MBTI personality test statistically robust and reliable?
2. Do the four dimensions of MBTI work independently, or are they connected in some way?
3. Do people with different MBTI types have different levels of activity on the internet?
4. Do significant differences exist in the interest preferences and behavioral patterns of different MBTI personality types on social networking sites?

4 Data Collection

1. "KPMIRU Questionnaires Data"

   The dataset was derived from a large-scale MBTI questionnaire response collection of 100,000 entries, shared on Kaggle by Pmenshih (https://www.kaggle.com/datasets/pmenshih/kpmiru-questionnaires-data).While each record originally includes binary responses to 63 forced-choice items (q1–q63) and accompanying numeric flags or timestamps (t1–t63), our analysis does not directly employ the individual item data. Instead, we leverage the two key derived outputs:

   - Four-letter MBTI type (psychotype), e.g. "ENTP" or "ISFJ," which provides a coarse segmentation of personality.

- Numeric dimension scores (e, i, s, n, t, f, j, p), each representing the total count of selections for that pole. These eight integer scores supply a continuous measure of preference intensity on each axis, which is essential for plotting score distributions, computing summary statistics, and conducting comparative dimension analyses across the sample.

2. "MBTI Personality Type Twitter Dataset"

   The MBTI Personality Type Twitter Dataset was originally compiled via the public Twitter API and later released on Kaggle by Mazlumi (https://www.kaggle.com/datasets/mazlumi/mbti-personality-type-twitter-dataset). The version employed here comprises 7,811 user records, each pairing a self-declared MBTI label (e.g., "ENTP") with the user's tweet content in raw form—hashtags, emojis, URLs, and @ (mention someone). This resource supports investigations of linguistic style, personality cues, and social-behavior patterns as reflected in self-identified MBTI categories.

   However, the dataset is markedly imbalanced: some personality types are over-represented while others appear only sparsely. Besides, this dataset is hard to clean since people's grammar on social media posts is quite casual, often including internet slang and non-standard abbreviations (e.g., "ur" means "you are"). Also, this dataset contains non-English text, making data cleaning even harder. Lastly, social-media text is notoriously noisy; even after extensive preprocessing, the corpus—about 7,800 tweet documents and replies, each averaging roughly 3,000 words—still falls short of capturing the full thematic breadth and linguistic diversity of the Twitter platform.

   By bringing together these two complementary datasets, we arrive at a balanced, evidence-based view of MBTI's strengths and limits.

```python
import pandas as pd
# Read data
data = pd.read_csv(r'E:\python\kpmi_ru_data.csv\kpmi_ru_data.csv')
print(data)
```

```
       q1  t1  q2  t2  q3  t3  q4  t4  q5  t5  ...  t63   e   i   s   n   t  \
0       1   8   2   8   2   8   1   8   1   8  ...    8  32  18  16  19  26
1       1   8   1   8   1   8   2   8   1   8  ...    8  28  15  27  18  23
2       1   8   1   8   1   8   1   8   1   8  ...    8  25  15  25  16  19
3       2   8   1   8   1   8   1   8   1   8  ...    8  30  15  25  14  33
4       2   8   2   8   1   8   1   8   1   8  ...    8  22  22  23  18  33
...    ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ...  ...  ..  ..  ..  ..  ..
99995   1   8   2   8   1   8   2   8   2   8  ...    8  32  18  27  16  33
99996   1   8   2   8   2   8   1   8   2   8  ...    8  20  26  37  12  37
99997   1   8   2   8   1   8   1   8   1   8  ...    8  22  18  23  19  26
99998   2   8   2   8   1   8   1   8   1   8  ...    8  22  18  21  23  33
99999   1   8   1   8   1   8   1   8   2   8  ...    8  38   9  39  12  37

        f   j   p psychotype
0      24  19  44       ENTP
1      30  35  29       ESFJ
2      35  38  23       ESFJ
3      27  38  23       ESTJ
4      13  29  35       ISTP
...    ..  ..  ..        ...
99995  19  13  52       ESTP
99996  13  29  29       ISTP
99997  22  29  38       ESTP
99998  19  32  29       ENTJ
99999  22  45  23       ESTJ

[100000 rows x 133 columns]
```

Figure 4.1: Reading KPMIRU Questionnaires Data

```python
import pandas as pd
# Read data
data = pd.read_csv(r"E:\python\twitter_MBTI.csv")
print(data)
```

```
      Unnamed: 0                                              text label
0              0  @Pericles216 @HierBeforeTheAC @Sachinettiyil T...  intj
1              1  @Hispanthicckk Being you makes you look cute||...  intj
2              2  @Alshymi Les balles sont réelles et sont tirée...  intj
3              3  I'm like entp but idiotic||Hey boy, do you wa...  intj
4              4  @kaeshurr1 Give it to @ZargarShanif ... He has...  intj
...          ...                                                ...   ...
7806        7806  @sobsjjun God,,pls take care 🥺|||@sobsjjun Hir...  intp
7807        7807  @Ignis_02 wow last time i got intp https://t.c...  intp
7808        7808  @akupilled A 100%||@akupilled That SOMEONE wi...  entp
7809        7809  If you’re #INTJ this one is for you | What is ...  infj
7810        7810  @harry__lambert @gucci hey can you dm me a pic...  istp

[7811 rows x 3 columns]
```

Figure 4.2: Reading MBTI Personality Type Twitter Dataset

As illustrated in the two Figures above, the combined Kaggle sources provide information on:

1. Self-reported MBTI types for each respondent
2. Raw Twitter posts and basic tweet metadata linked to those MBTI labels
3. Demographic and psychometric questionnaire answers (KPMIRU survey)
4. Behavioral metrics such as posting frequency and topic keywords extracted from the tweets

5 MBTI in Measurement

5.1 Data Cleaning and Pre-processing

5.1.1 Handling Missing Values

1. We use `dropna()` to eliminate any records with missing or null entries. Fortunately, the dataset had no missing psychotype labels or scoring data.

```python
# Delete null values
df_clean = df.dropna()
```

5.1.2 Outlier Detection and Correction

1. Outliers in numeric scores were identified using the **IQR (interquartile range) method**.
2. For each numeric column, we computed Q1, Q3. IQR = Q3 - Q1 represent the middle 50% of the data distribution. The `lower_bound` and `upper_bound` represent the boundaries of the normal range under the Interquartile Range Rule (IQR). And then we replaced values outside [Q1 − 1.5×IQR, Q3 + 1.5×IQR] with the column median.
3. The median substitution method mainly serves to stabilize the overall distribution and avoid the influence of extreme values.

```python
# Select all columns in df_clean with numeric data types (either int or float) in preparation for outlier processing.
numeric_cols = df_clean.select_dtypes(include='number').columns

# Replace the outliers (utilizing the Interquartile Range (IQR) rule) with the median value of the column.
df_processed = df_clean.copy()

for col in numeric_cols:
    Q1 = df_processed[col].quantile(0.25)
    Q3 = df_processed[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    median_value = df_processed[col].median()

    # Replace outliers
    df_processed[col] = df_processed[col].apply(lambda x: median_value if x < lower_bound or x > upper_bound else x)
```

5.1.3 Export the cleaned data

After completing the outlier replacement and data cleaning process, we used df.info() to verify the integrity and structure of the cleaned dataset. The cleaned DataFrame was then exported using the `to_csv()` method, which saved it as `kpmi_ru_data(Cleaned).csv` for downstream analysis. The `index=False` parameter ensured that row indices were not written to the CSV file.

```python
print(df_processed.info())
print(df_processed)
df_processed.to_csv(r'E:\python\kpmi_ru_data(Cleaned).csv', index = False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Columns: 133 entries, q1 to psychotype
dtypes: float64(75), int64(57), object(1)
memory usage: 101.5+ MB
None
       q1   t1  q2   t2  q3   t3  q4   t4  q5   t5  ...  t63     e     i  \
0       1  8.0   2  8.0   2  8.0   1  8.0   1  8.0  ...  8.0  32.0  18.0   
1       1  8.0   1  8.0   1  8.0   2  8.0   1  8.0  ...  8.0  28.0  15.0   
2       1  8.0   1  8.0   1  8.0   1  8.0   1  8.0  ...  8.0  25.0  15.0   
3       2  8.0   1  8.0   1  8.0   1  8.0   1  8.0  ...  8.0  30.0  15.0   
4       2  8.0   2  8.0   1  8.0   1  8.0   1  8.0  ...  8.0  22.0  22.0   
...    ..  ...  ..  ...  ..  ...  ..  ...  ..  ...  ...  ...   ...   ...   
99995   1  8.0   2  8.0   1  8.0   2  8.0   2  8.0  ...  8.0  32.0  18.0   
99996   1  8.0   2  8.0   2  8.0   1  8.0   2  8.0  ...  8.0  20.0  26.0   
99997   1  8.0   2  8.0   1  8.0   1  8.0   1  8.0  ...  8.0  22.0  18.0   
99998   2  8.0   2  8.0   1  8.0   1  8.0   1  8.0  ...  8.0  22.0  18.0   
99999   1  8.0   1  8.0   1  8.0   1  8.0   2  8.0  ...  8.0  38.0   9.0   

          s     n     t     f     j     p psychotype  
0      16.0  19.0  26.0  24.0  19.0  44.0       ENTP  
1      27.0  18.0  23.0  30.0  35.0  29.0       ESFJ  
2      25.0  16.0  19.0  35.0  38.0  23.0       ESFJ  
3      25.0  14.0  33.0  27.0  38.0  23.0       ESTJ  
4      23.0  18.0  33.0  13.0  29.0  35.0       ISTP  
...     ...   ...   ...   ...   ...   ...        ...  
99995  27.0  16.0  33.0  19.0  13.0  52.0       ESTP  
99996  37.0  12.0  37.0  13.0  29.0  29.0       ISTP  
99997  23.0  19.0  26.0  22.0  29.0  38.0       ESTP  
99998  21.0  23.0  33.0  19.0  32.0  29.0       ENTJ  
99999  39.0  12.0  37.0  22.0  45.0  23.0       ESTJ  

[100000 rows x 133 columns]
```

The data in the figure 3.1 is the data after our data cleaning.

5.2 Model Building and Evaluation

Chi-square test is an on parametric statistical test to determine if the two or more classifications of the samples are independent or not (Zibran, 2007). We all know that MBTI has four dimensions (Extraversion–Introversion (E–I), Sensing–Intuition (S–N), Thinking–Feeling (T–F) and Judging–Perceiving (J–P)). But whether or not these four dimensions interrelated or independent of each other stays unknown. For explanation, let's consider the data presented in Figure 3.1 which comprising 100 000 respondents, providing information on the scoring fields of the four dimensions of MBTI. In order to find the answer, we use the chi-square test.

Each respondent's type label (e.g., ENTP) was decomposed into its four constituent letters, and the sample was cross-classified into a $2 \times 2 \times 2 \times 2$ contingency table (16 cells)(shown in figure 3.2)

```
import pandas as pd
import numpy as np
from scipy.stats import chi2

# 1) Read in the cleaned CSV.
FILE_PATH = r"E:\python\kpmi_ru_data(Cleaned).csv"
df = pd.read_csv(FILE_PATH)

# 2) Extract the columns of four dimensions .IE, SN, TF, JP
mbti = df['psychotype'].astype(str).str.upper().dropna()
df_dims = pd.DataFrame({
    'IE': mbti.str[0],    # E  I
    'SN': mbti.str[1],    # S  N
    'TF': mbti.str[2],    # T  F
    'JP': mbti.str[3]     # J  P
})

# 3)Construct a 2×2×2×2 contingency table: Obtain the observed frequencies
  of 16 cells. counts[i,j,k,l]
import pandas as pd
import numpy as np

# Define dimension labels
index = pd.MultiIndex.from_product(
    [['E','I'], ['S','N'], ['T','F'], ['J','P']],
    names=['IE','SN','TF','JP']
)

# Build a DataFrame, using the values of counts as columns.
counts_df = pd.DataFrame({
    'Count': counts.flatten()
}, index=index)

print(counts_df.head(10))
```

Then we calculate the marginal distribution (the distribution of each dimension separately). For example, N_IE = [count(E), count(I)](shown in **??**)

```
# Calculate marginal distribution and total number
N_IE = counts.sum(axis=(1,2,3))
N_SN = counts.sum(axis=(0,2,3))
N_TF = counts.sum(axis=(0,1,3))
N_JP = counts.sum(axis=(0,1,2))
N = counts.sum()


marginals = pd.DataFrame({
    'Dimension': ['IE', 'IE', 'SN', 'SN', 'TF', 'TF', 'JP', 'JP'],
    'Category': ['E', 'I', 'S', 'N', 'T', 'F', 'J', 'P'],
    'Count': np.concatenate([N_IE, N_SN, N_TF, N_JP])
})


print(marginals)
```

Under the null hypothesis of mutual independence, the expected frequency in each cell was computed as the product of the four one-dimensional marginal distributions multiplied by the sample size(shown in **??**).

```
#Calculate the expected frequency E[i,j,k,l] of each cell based on the
  completely independent assumption.
E = np.zeros_like(counts, dtype=float)
for i in range(2):
    for j in range(2):
        for k in range(2):
            for l in range(2):
                E[i,j,k,l] = (
                    (N_IE[i]/N) *
                    (N_SN[j]/N) *
                    (N_TF[k]/N) *
                    (N_JP[l]/N)
                ) * N

index = pd.MultiIndex.from_product(
    [['E', 'I'], ['S', 'N'], ['T', 'F'], ['J', 'P']],
    names=['IE', 'SN', 'TF', 'JP']
```

```
)

expected_df = pd.DataFrame({
    'Expected_Count': E.flatten()
}, index=index).round(2)

print(expected_df)
```

Now, it's time to calculate the Pearson chi-square statistic, while O is observed value and E is expected value.

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

```
chi2_stat = ((counts - E)**2 / E).sum()
chi2_stat
```

With this code, we obtain the Pearson chi-square statistic, which is 1342.93063615322.

Finally, with $\chi^2 = 1342.93$ and df = 11,we can compute the right - tailed probability corresponding to the chi - square statistic under 10 degrees of freedom, which is 0. Because the p-value falls far below the conventional $\alpha = 0.05$ threshold, the null hypothesis is decisively rejected: the observed joint distribution of MBTI preferences deviates dramatically from what would be expected if the four indices varied independently. In practical terms, substantial associations exist among the E–I, S–N, T–F and J–P scales, corroborating earlier psychometric critiques that the MBTI dimensions are not orthogonal factors but overlap to a non-trivial extent.

Actually, some researchers had also proven that the four dimensions of MBTI are not independent. Fleenor (1997) investigated the intercorrelations among MBTI continuous scores and found that while most dimension pairs demonstrated relatively low correlations, the correlation between the Sensing–Intuition (S-N) and Judging–Perceiving (J-P) scales was notably higher. Specifically, the study reported a correlation coefficient of r = 0.41 between S-N and J-P, indicating a moderate positive relationship. This finding has been replicated in other research and suggests that individuals who prefer intuition are more likely to also prefer perceiving. As such, the assumption of strict statistical independence between MBTI preference axes, particularly between S-N and J-P, may not hold, which is in line with the findings of our study.

```
df_val = 10
p_val = 1 - chi2.cdf(chi2_stat, df_val)
print(p_val)
```

```
if p_val < 0.05:
    print("Conclusion: Reject  H -- The four dimensions are not completely
      independent.")
else:
    print("Conclusion:  H cannot be rejected - the four dimensions can be
      regarded as independent.")
```

5.3 Exploratory Data Analysis (EDA)

5.3.1 MBTI Type Frequency

```
# 2) Frequency bar chart
plt.figure(figsize=(10,4))
df['psychotype'].value_counts().plot(kind='bar')
plt.title('Frequency of 16 MBTI Types')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

A frequency bar chart of the 16 MBTI types was generated using the value_counts() function on the psychotype column. The result was plotted using matplotlib and is shown in fig. 5.1.

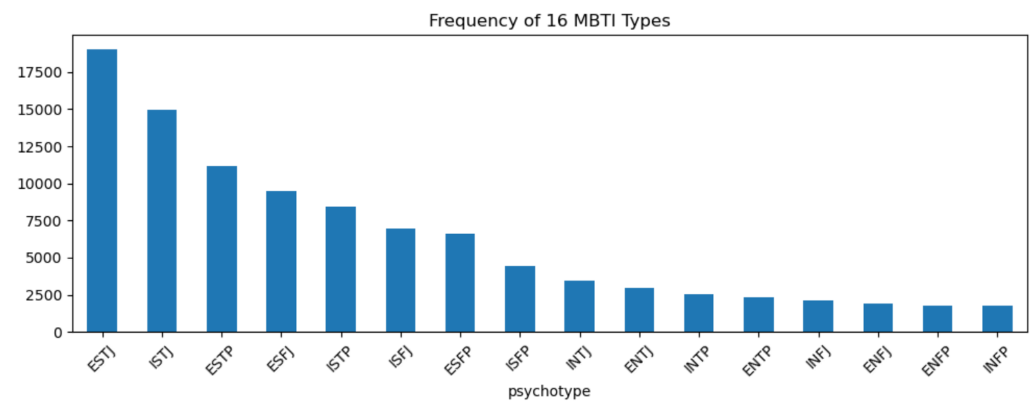


Figure 5.1: Frequency of 16 MBTI Types

As seen in fig. 5.1, the frequency of 16 MBTI types has been shown in a descending order. The type ESTJ ranks first, which shows that, based on our dataset, the proportion of ESTJ people is the highest. Conversely, the proportion of INFP people is the lowest. Additionally, it can be observed that the _S_J people rankings are relatively high in frequency, as the _NF_ people rankings are relatively low.

Also, fig. 5.1 exhibits severe right oblique imbalance distribution, as the number of ESTJ people is nearly 9 times as many as the number of INFP people. This phenomenon may be attributed to our database being originally from Kaggle, and certain MBTI type tends to participate in such investigations.

This contrasts with national MBTI distribution statistics reported in the MBTI® Manual, where ISFJ and ESFJ were found to be the most common types among U.S. adults (Myers et al., 1998), suggesting that our dataset, to a certain extent, fits the population-level trends. In distinct regions, the regional differences may influence MBTI type distributions in specific rankings.

5.3.2 Distribution of MBTI Dimension Scores

```
# 3) Histogram of scores in each dimension
score_cols = list('eisntfjp')
df[score_cols].hist(bins=20, figsize=(12,6))
plt.tight_layout()
plt.show()
```

Histograms of scores in each MBTI dimension were generated using pandas.DataFrame.hist and visualized with matplotlib as shown in fig. 5.2.

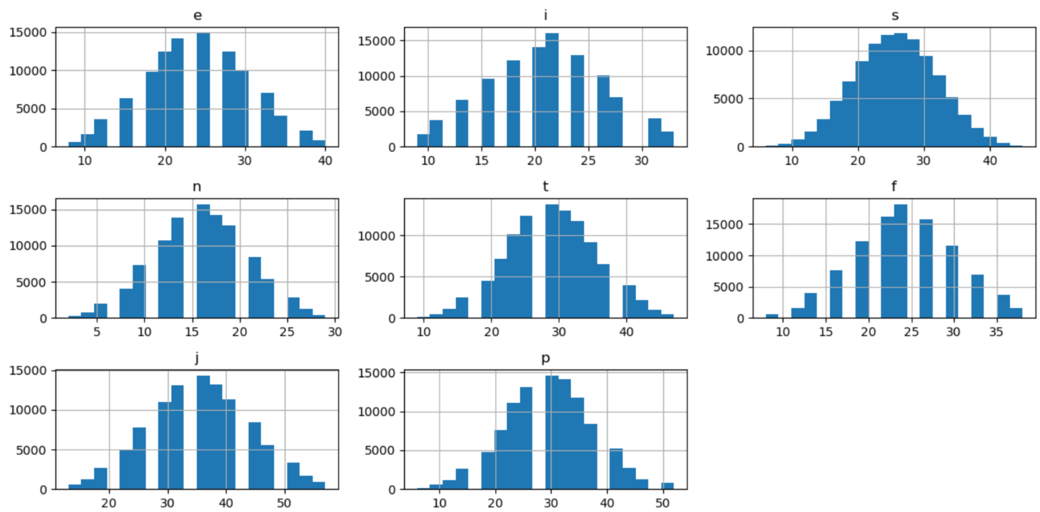


Figure 5.2: Histogram of MBTI Dimension Scores

As seen in fig. 5.2, from the score concentration trend, the eight MBTI poles vary in concentration. For example, E, S, and T scores are generally spread from 10 to 40, while N and I scores are more narrowly concentrated between 5 to 30. And in detail, in the dimension of P and T, exhibit higher scores, which shows the preference toward Thinking and Perceiving. Compared to them, in the dimension of N and I, they exhibit lower scores, which indicates most people are more inclined toward Extraversion and Sensing. From the data obtained from these diagrams, we can observe that most samples in these dimensions show distributional asymmetries, as not evenly distributed. Particularly for the high values of J and low values of N, this result matches the observation at 4.1.1 that _S_J types dominate the database.

5.3.3 Descriptive Statistics of MBTI Dimensions

```python
# 4) Describe statistics (mean, std, quartile, skewness, kurtosis)
stats = df[score_cols].describe().T
stats['skew']  = df[score_cols].skew()
stats['kurt']  = df[score_cols].kurt()
print(stats.round(2))
```

Descriptive statistics of each MBTI dimension were computed and consolidated into a single summary table, including skewness and kurtosis, to facilitate score distribution analysis shown in fig. 5.3.

	count	mean	std	min	25%	50%	75%	max	skew	kurt
e	100000.0	24.00	6.52	8.0	20.0	25.0	28.0	40.0	0.02	-0.38
i	100000.0	21.00	5.43	9.0	18.0	22.0	24.0	33.0	-0.06	-0.44
s	100000.0	26.19	6.65	6.0	21.0	27.0	31.0	45.0	-0.09	-0.19
n	100000.0	16.12	4.71	2.0	12.0	16.0	19.0	29.0	-0.11	-0.05
t	100000.0	28.89	6.64	9.0	23.0	28.0	33.0	47.0	0.00	-0.25
f	100000.0	24.05	5.92	8.0	19.0	24.0	27.0	38.0	-0.08	-0.25
j	100000.0	35.41	8.80	13.0	29.0	35.0	41.0	57.0	0.01	-0.31
p	100000.0	29.34	7.93	6.0	23.0	29.0	35.0	52.0	0.05	-0.19

Figure 5.3: Descriptive Statistics of MBTI Dimension Scores

Based on Table 4.1.3, it can be observed that the dimensions of I, S, N, and F showcase a notable positive skew distribution, as the dimension of N, with a value of -0.11, demonstrates left skewness most. This phenomenon tends to show that most people are more likely to be Sensing(S) and so on. And in contrast, the dimension of P with the value of +0.05 exhibits right skewness, implying a preference towards lower values, showing that Judging(J) dominates more in this database. Results of these also support the conclusion in Figure 4.1.1 that _S_J people occupy the majority.

And from the kurtosis value, it can be seen that most values are around 0 and negative, which suggests that all the dimension is distributed platykurtic. With relatively concentrated values, this exhibits the loss of extreme outliers. Also, the maximum value and minimum value that all dimensions have can be used in the form of Max-Min, which demonstrates that J and P dimensions have the largest difference, indicating that individual differences between them are the strongest.

		S		N	
		T n(%)	F n(%)	T n(%)	F n(%)
E	J	ESTJ	ESFJ	ENTJ	ENFJ
		30 (12.2)	32 (13.1)	12 (4.9)	11 (4.5)
	P	ESTP	ESFP	ENTP	ENFP
		12 (4.9)	30 (12.2)	14 (5.7)	20 (8.2)
I	J	ISTJ	ISFJ	INTJ	INFJ
		29 (11.8)	21 (8.6)	2 (0.8)	2 (0.8)
	P	ISTP	ISFP	INTP	INFP
		12 (4.9)	10 (4.1)	3 (1.2)	5 (2.0)

Figure 5.4: MBTI Distribution of 16 Personality Types

Figure 4.1.3 MBTI Distribution of 16 Personality Types from Jang and Kim (2014), supporting the dominance of _S_J types observed in our dataset.

5.3.4 Correlation Analysis Between MBTI Dimensions

```
# 5) Correlation matrix heat map
plt.figure(figsize=(6,5))
sns.heatmap(df[score_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation between dimension scores')
plt.tight_layout()
plt.show()
```

A Pearson correlation heatmap was constructed to visualize linear relationships among MBTI dimension scores, as shown in fig. 5.5.

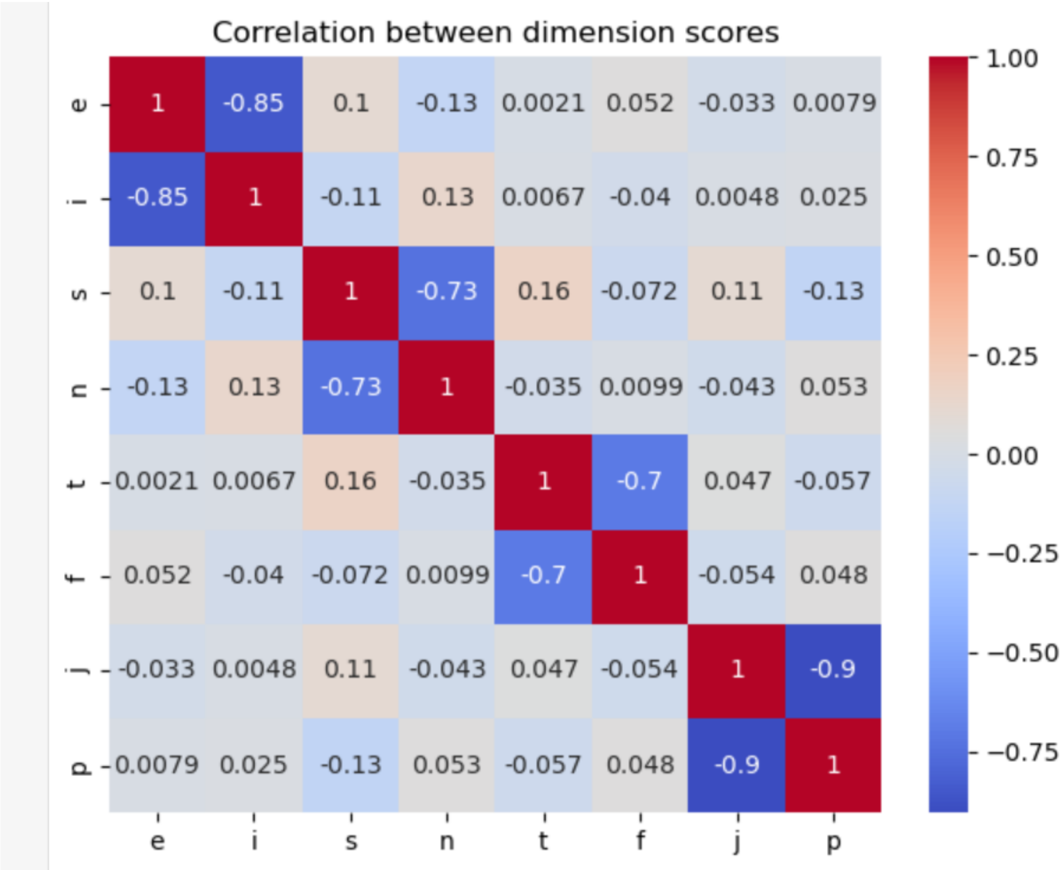


Figure 5.5: Correlation Heatmap of MBTI Dimension Scores

As seen in fig. 5.5, the correlation between the opposite dimensions, like E vs I, S vs N, T vs F, and J vs P. All of them are strongly negative, which is consistent with the design concept of MBTI, as the two poles of each dimension are opposed. As a result, the MBTI method has its rationality. This result aligns with (Li et al., 2024), who also observed that "the correlations between the four axes are generally weak, indicating that the personality traits on each axis are relatively independent" (p. 12).

Furthermore, in fig. 5.5, the correlation between other dimension pairs is low (nearly 0). This phenomenon supports that the dimensions are seemingly independent. While the correlation between S and T is +0.16, a bit larger than others, it indicates that someone who is Sensing(S) tends to be Thinking(T). In contrast, the correlation between S and P is -0.13, which indicates that those who are Sensing(S) tend to be Judging(J).

6 MBTI in Manifestation

In the previous section, we examined the statistical characteristics of MBTI. In this section, we turn to the manifestation level by investigating how different MBTI types vary in their topic preferences and expressive patterns on Twitter.

6.1 Topic Preferences of MBTI Personalities

To analyze the topic preference of different MBTI personalities, we adopt **LDA modeling**.

Latent Dirichlet Allocation (LDA) is a generative probabilistic model for uncovering hidden "topics" in a large collection of documents. It assumes each document is composed of a mixture of topics, and each topic is represented by a distribution over words. For example, a document might be 30% about "technology" and 70% about "health", with each topic associated with its own common vocabulary.

In the LDA framework, the generation of a document is imagined as a two-step process:

1. a topic distribution is assigned to the document.
2. For each word in the document:
    - A topic is randomly chosen based on the document's topic distribution.
    - A word is selected from the vocabulary of that topic.

During model training, this generative process is reversed: LDA infers the underlying topic structure based on the observed words in the documents. It estimates:

- the topic proportions for each document
- the most representative words for each topic

The specific LDA process of our project will be discussed in detail. However, LDA model has a extremely high requirement for text data cleaning, since LDA relies on word-frequency patterns to infer hidden topics. It's important to remove noise, such as words that are meaningless or unrepresentative, so that the model can produce reliable topics. Thus, we need to make sure our dataset is clean and ready for LDA modeling.

6.1.1 Data Cleaning

Because of the nature of the project, we must work with a dataset that contains a large volume of text. The raw dataset we obtained suffers from the following issues:

1. Text granularity / structure: In each row, the posts field actually concatenates 50 posts from a single user, separated by "|||".

2. Dirty characters and encoding: The data contain URLs, HTML fragments, escape sequences, and leading single quotation marks.

3. Length and memory footprint: Each row is about 7 k characters on average, with some rows exceeding 10 k characters.

These problems prevent the dataset from being used directly for analysis, so we must clean it first. Importing Modules and the Dataset

1. Import the required library

```python
import pandas as pd
```

This line imports the pandas library under the alias pd. Pandas is a powerful tool for data analysis and manipulation, especially well suited for reading and processing structured data such as CSV files.

2. Import N analyze data p1 LP related modules

```python
import re, string
from typing import List
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk import pos_tag
from nltk.stem import WordNetLemmatizer
```

These statements import several NLP utilities:

- re, string: regular expression and character utilities for text processing.
- typing.List: type annotation for list data structures.
- The nltk suite: tokenisation (word_tokenize), stop word handling (stopwords), part of speech tagging (pos_tag), lemmatisation (WordNetLemmatizer), and more.

```python
my_nltk_path = "Data"
nltk.data.path.append(my_nltk_path)
```

This sets the local NLTK resource path to the Data directory and appends it to the search path so that NLTK can load the required resources in the local environment.

```python
import textstat
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

- textstat: assesses text readability.
- SentimentIntensityAnalyzer: VADER's sentiment analysis tool for assessing the affective polarity of text.

3. Import modules for reading and writing external files

```python
import json
import pickle
import copy
```

- json: parsing and serialising JSON data.
- pickle: serialising and deserialising Python objects.
- copy: making shallow and deep copies of objects.

4. Load the FastText language identification model

```python
import fasttext
lang_model = fasttext.load_model("lid.176.bin")
```

This loads Facebook's FastText model lid.176.bin, which recognises 176 languages and returns language codes (e.g. "en", "zh", "es"). It is essential when pre processing multilingual data—especially when we need to standardise text to English via translation.

5. Load a local large language model (LLM) and specify the model version

```python
import ollama
llm = "llama3.1:8b"
```

ollama provides an interface for running LLMs locally. Here we call the LLaMA 3 model with 8 billion parameters, a lightweight model used for on premise translation to improve privacy and efficiency when translating large corpora offline.

6. Bring in the tqdm progress bar and enable Pandas integration

```python
from tqdm.auto import tqdm
tqdm.pandas()
```

TQDM is a popular Python progress bar library. tqdm.auto selects the most suitable display backend automatically (Jupyter friendly). Calling tqdm.pandas() enables progress bars for DataFrame.apply() and Series.apply() operations, giving real time feedback during long running tasks.

7. Define an average value helper

```
def ave(l):
    return sum(l) / len(l)
```

A simple function ave is defined to calculate the average of a list of numeric values.

8. Define the list of MBTI personality types

```
MBTI_types = [
    "ISTJ", "ISFJ", "INFJ", "INTJ",
    "ISTP", "ISFP", "INFP", "INTP",
    "ESTP", "ESFP", "ENFP", "ENTP",
    "ESTJ", "ESFJ", "ENFJ", "ENTJ"
]
```

A list named MBTI_types is defined, containing all 16 Myers-Briggs personality types, to serve as the classification basis in subsequent text or label processing.

9. Read the data and perform an initial split

```
raw_data = pd.read_csv("Data\\twitter_MBTI.csv", encoding="utf-8")
raw_data.drop(columns="Unnamed: 0", inplace=True)
raw_data.columns = ["posts", "type"]

for i in raw_data.index:
    temp = raw_data.loc[i, "posts"]
    temp = temp.split("|||")
    raw_data.loc[i, "posts"] = temp
```

First, the CSV file twitter_MBTIċsv is loaded into a DataFrame. The automatically added index column (”Unnamed: 0”) is removed, and the columns are renamed to standardise their labels. Each row’s posts field is then split on the delimiter ”|||”, converting the concatenated string into a list of separate posts for subsequent sentence level analysis.

10. Load and merge the custom stop word list

```python
from custom_stopwords import custom_stopwords
stop_words.update(custom_stopwords)
```

custom_stopwords is a task specific collection of terms—such as social media jargon (”lol”, ”omg”, ”http”), MBTI irrelevant proper nouns, or noise tokens—that should be excluded. Merging it into stop_words ensures these words are filtered out during cleaning.

Methods for Data Cleaning

1. Loading the contraction mapping

```python
with open(file="contractions.json", mode='r', encoding='utf-8') as f:
    contractions_map = json.load(f)
```

contractions.json maps common English contractions to their full forms (e.g. ”don't” → ”do not”), facilitating normalisation of user-generated content.

2. Constructor __init__

```python
def __init__(self, source=raw_data):
    self.data = source
```

Initialises the object with a DataFrame that contains a posts column in which each entry is a list of sentences.

3. Remove user mentions and hashtags

```python
def remove_mention_and_tag(self):
    def process_removal(post):
        post_without_mention=[]
        for sentence in post:
            # Use re to scan and substitute
            post_without_mention.append(
                re.sub(
                    pattern=r'@\w+|#\w+',
                    repl=' ',
                    string=sentence
                )
            )
        return post_without_mention
    self.data["posts"]=self.data["posts"].apply(process_removal)
```

This method defines an inner function `process_removal` that scans each sentence in a post with the regular expression `r'@\w+|#\w+'` and replaces every mention or hashtag with a space, thereby stripping social media tags while preserving sentence order. The `apply()` call performs this operation on the entire `posts` column.

4. Delete URL hyperlinks

```python
def remove_url(self):
    def process_remove_url(post):
        post_without_url = []
        for sentence in post:
            post_without_url.append(
                re.sub(
                    pattern=r'http\S+|www\S+|https\S+',
                    repl='',
                    string=sentence,
                    flags=re.MULTILINE
                )
            )
        return post_without_url
    self.data["posts"] = self.data["posts"].apply(process_remove_url)
```

The method removes any web page address (e.g. http://…, www.…), preventing links from contaminating downstream text analysis.

5. Strip emoji characters

```python
    def remove_emoji(self):
        def process_remove_emoji(post):
            post_without_emoji=[]
            for sentence in post:
                # Use re to scan and substitute
                emoji_pattern=re.compile(
        "["
        "\U0001F600-\U0001F64F"  # Emoticons
        "\U0001F300-\U0001F5FF"  # Miscellaneous Symbols and Pictographs
        "\U0001F680-\U0001F6FF"  # Transport and Map Symbols
        "\U0001F1E0-\U0001F1FF"  # Flags (iOS)
        "\U00002702-\U000027B0"  # Dingbats
```

```
        "\U000024C2-\U0001F251"  # Enclosed Characters, etc.
        "\U0001f926-\U0001f937"  # Supplemental Symbols and
 Pictographs
        "\U00010000-\U0010ffff"  # Broader range for some less common
 emojis
        "]+", flags=re.UNICODE
                )
```

The regular expression spans multiple Unicode ranges—covering most common and uncommon emojis such as facial expressions, gestures, vehicles, map icons, a wide variety of regional flags and pictographic symbols, and even supplementary characters beyond the Basic Multilingual Plane (BMP)—with flags=re.UNICODE enabled to ensure all Unicode characters are recognized.

```
                post_without_emoji.append(
                    emoji_pattern.sub(
                        repl=' ',
                        string=sentence
                    )
                )
            return post_without_emoji
        self.data["posts"]=self.data["posts"].apply(
 process_remove_emoji)
```

For each sentence, the regex matches emojis and replaces every matched character with a single space (' '), preventing token concatenation noise while maintaining readability and analyzability; the process_remove_emoji function is then applied to the entire posts column via .apply(), bulk-removing emojis from every post and overwriting the original data column with the cleaned results.

6. Expand English contractions

Static method text_expand()

```
@staticmethod
def text_expand(original_string, contraction_mapping=contractions_map)
 :
```

Defines a static method `text_expand()` that takes a single string (a sentence) as input and, by default, uses the pre-loaded dictionary of contractions `contractions_map` to convert contractions to their full forms.

```
contractions_pattern = re.compile('({})'.format('|'.join(
  contraction_mapping.keys())), flags=re.IGNORECASE | re.DOTALL)
```

This statement concatenates all the keys (i.e., every contraction) in the mapping dictionary into a regular expression pattern so that any occurrence of a contraction in the text can be captured. The flag `re.IGNORECASE` makes the match case insensitive, and `re.DOTALL` allows the pattern to span across line breaks.

```
def text_mapping(text_matched):
    old_text = text_matched.group(0)
    new_text = contraction_mapping.get(old_text.lower())
    if not new_text:
        new_text = contraction_mapping.get(old_text)
        if not new_text:
            return old_text
    return new_text
```

The inner function `text_mapping()` replaces each matched contraction with its full form. It first tries to match the lowercase version; if that fails, it tries the original case. If a replacement still cannot be found, it returns the original text unchanged, preventing information loss.

```
expanded_string = contractions_pattern.sub(repl=lambda m: text_mapping
  (m), string=original_string)
return expanded_string
```

Using `re.sub()`, every match in the original string is replaced, and the expanded string is finally returned.

```
def expand_contractions(self):
    def process_expand_contractions(original_list):
        for idx in range(len(original_list)):
            original_list[idx] = Data_to_Clean.text_expand(
              original_list[idx])
        return original_list
    self.data["posts"] = self.data["posts"].apply(lambda x:
      process_expand_contractions(x))
```

This method batch-processes every `posts` entry in the DataFrame. Each `posts` entry is a list of strings, representing multiple sentences written by a user. The nested function process_expand_contractions() calls the static method text_expand() on every sentence in the list, expanding all contractions to their full forms. Finally, apply() is used to apply the processing to the entire column, achieving large-scale contraction expansion. This greatly improves the consistency and semantic clarity of the data, which benefits subsequent tokenization, vectorization, and modelling stages.

```
def tolower(self):
    def process_tolower(post):
        return [sentence.lower() for sentence in post]
    self.data["posts"] = self.data["posts"].apply(process_tolower)
```

Uniform lower casing prevents word-frequency fragmentation due to case differences.

```
def remove_punct(self):
    def process_remove_punct(post):
        post_without_punct = []
        for sentence in post:
            post_without_punct.append(
                re.sub(pattern=r'[^a-zA-Z\s]', repl=' ', string=
                  sentence)
            )
        return post_without_punct
    self.data["posts"] = self.data["posts"].apply(process_remove_punct
      )
```

Non-alphabetic characters are replaced by spaces, simplifying the text for language modelling.

```
def remove_whitespace(self):
    def process_remove_whitespace(post):
        return [sentence for sentence in post if sentence.strip()]
    self.data["posts"] = self.data["posts"].apply(
      process_remove_whitespace)
```

Sentences that are empty or contain only whitespace are discarded.

```
def totokens(self):
    def process_totokens(post):
        post_totokens = []
        for sentence in post:
            tokens = word_tokenize(sentence)
```

```
            post_totokens.append(tokens)
        return post_totokens
    self.data["posts"] = self.data["posts"].apply(process_totokens)
```

`word_tokenize()` splits each cleaned sentence into a list of tokens, providing the basic units for further NLP tasks.

```
def remove_stopwords(self):
    def process_remove_stopwords(post):
        stop_words = set(stopwords.words("english"))
        filtered_post = []
        for sentence in post:
            filtered_sentence = [word for word in sentence if word not in stop_words]
            filtered_post.append(filtered_sentence)
        return filtered_post
    self.data["posts"] = self.data["posts"].apply(process_remove_stopwords)
```

Common high-frequency, low-information words are filtered out to emphasise lexical content relevant for modelling.

```
def post_lemmatize(self):
    def process_lemmatize(post):
        def get_wordnet_postag(old_postag):
            if old_postag.startswith('J'): return wordnet.ADJ
            elif old_postag.startswith('V'): return wordnet.VERB
            elif old_postag.startswith('N'): return wordnet.NOUN
            elif old_postag.startswith('R'): return wordnet.ADV
            else: return wordnet.NOUN

        lemmatizer = WordNetLemmatizer()
        lemmatized_post = []
        for tokens in post:
            tagged_tokens = pos_tag(tokens)
            lemmatized_tokens = [lemmatizer.lemmatize(word, get_wordnet_postag(tag)) for word, tag in tagged_tokens]
            lemmatized_post.append(lemmatized_tokens)
        return lemmatized_post
    self.data["posts"] = self.data["posts"].apply(process_lemmatize)
```

Words are reduced to their dictionary forms based on POS tags, decreasing dimensionality and unifying inflected variants.

```
def drop_empty(self):
        def process_drop(post):
            result=[sentence for sentence in post if sentence!=[]]
            return result
        self.data["posts"]=self.data["posts"].apply(process_drop)
```

After cleaning, residual empty sentences are removed to maintain structural integrity.

1. **Automatically Detect Non-English Sentences**

```
def locate_str_to_translate(self):
    result = []
    for i in tqdm(self.data.index,
                  desc="Locating strings of other languages..."):
        for j in range(len(self.data.loc[i, "posts"])):
            sentence = self.data.loc[i, "posts"][j]

            # Only check sentences longer than eight words
            if len(sentence.split()) > 8:
                lang_prediction = lang_model.predict(
                    re.sub(r'\s+', ' ', sentence).strip(),
                    k=1                              # top-1 prediction
                )

                # Keep if not English & confidence > 0.98
                if (lang_prediction[0][0] != "__label__en"
                        and lang_prediction[0][1] > 0.98):
                    result.append(
                        (i, j,
                         re.sub(pattern=r"__\w+__",
                                repl='',
                                string=lang_prediction[0][0]))
                    )

    self.locations = result
    with open(f"Data/{self.basic_identities['type']}
      _translate_location.pkl",
            "wb") as f:
```

```
        pickle.dump(result, f)
```

- **Outer loop**: iterates over the entire DataFrame index, visiting each post sequentially.
- **Inner loop**: traverses every sentence inside the current post (a list of sentences).
- **tqdm** renders a progress bar for large-scale datasets.
- A lower bound of > 8 words avoids language-detection errors on extremely short sentences.
- The FastText model returns a tuple `('__label__xx', confidence)`; only non-English predictions with confidence > `0.98` are stored.
- Results are saved both in `self.locations` and pickled to disk for later translation.

2. **Translate with a Local LLM**

```python
def translate_str(self):
    for coord in tqdm(self.locations,
                      desc="Translating string into English..."):
        sentence = self.data.loc[coord[0], "posts"][coord[1]]

        self.data.loc[coord[0], "posts"][coord[1]] = \
            ollama.generate(
                model=llm,
                prompt=(
                    f"Translate from {coord[2]} to English: \"{sentence}\"\n"
                    f"Output ONLY the translated text."
                )
            )['response']
```

Each non-English sentence identified in Step 8 is translated *in place* by a locally hosted LLM (e.g. `llama3.1:8b`). The prompt specifies the source language and requests only the English rendition, preserving post order and structure.

3. **Filter Non-English Sentences by Language ID**

```python
def process_drop(post, level=0.98):
    filtered_post = []
```

```
    for sentence in post:
        norm = re.sub(r"\s+", " ", sentence)

        # Keep very short sentences to avoid false negatives
        if len(norm.split()) < 6:
            filtered_post.append(norm)
            continue

        lang = lang_model.predict(norm)
        if lang[0][0] == '__label__en' and lang[0][1] > level:
            filtered_post.append(norm)
        else:
            # Retry in lowercase for robustness
            lang = lang_model.predict(norm.lower())
            if lang[0][0] == '__label__en' and lang[0][1] > level:
                filtered_post.append(norm)
    return filtered_post

self.data["posts"] = self.data["posts"].apply(process_drop)
```

- Filters out non-English sentences without translating them.
- Sentences shorter than six words are retained to reduce false negatives.
- Longer sentences must pass an English-confidence threshold (`level`, default `0.98`).
- A second check on the lowercase version mitigates errors from mixed casing or acronyms.

Create a data processing method that encompasses the entire workflow

1. **Initialize the Data Object & Remove URLs**

```
data = Data_to_Analyze(type=TYPE)
data.remove_url()
data.remove_mention_and_tag()
```

- Instantiate a `Data_to_Analyze` object to load the raw dataset for the specified personality type TYPE.
- Call `remove_url()` and `remove_mention_and_tag()` to strip out hyperlinks, user mentions (`@username`), and topic hashtags (`#topic`), eliminating social-media noise.

2. **Extract Structured Text Features (Before Cleaning)**

```
data.get_sentence_quantity()
data.get_word_count()
data.get_upper_ratio()
data.get_readability()
data.get_vader_score()
```

- `get_sentence_quantity()` – counts the number of sentences in each post.
- `get_word_count()` – totals the words in each post.
- `get_upper_ratio()` – computes the ratio of uppercase letters to all alphabetic characters.
- `get_readability()` – evaluates readability using two `textstat` metrics: Flesch Reading Ease and Gunning Fog Index.
- `get_vader_score()` – applies the VADER model for sentiment analysis, yielding composite sentiment scores.

Important: Run these functions *before* any cleaning steps such as contraction expansion or case conversion, since those operations alter the statistics.

3. **Continue Text-Cleaning Tasks**

```
data.remove_emoji()
data.remove_whitespace()
data.drop_non_english(0.75)
data.expand_contractions()
data.tolower()
data.remove_punct()
data.remove_whitespace()
data.totokens()
```

- `remove_emoji()` – removes emoji and other non-text graphical elements.
- `remove_whitespace()` (first pass) – performs an initial whitespace cleanup.
- `drop_non_english(0.75)` – uses fastText to detect language and drops sentences that are not English when confidence is below `0.75`.

- `expand_contractions()` – expands English contractions (e.g., “don’t” → “do not”).
- `tolower()` – converts all text to lowercase, reducing word-form variability.
- `remove_punct()` – strips punctuation to facilitate vectorization or model processing.
- `remove_whitespace()` (second pass) – clears residual extra spaces from earlier steps.
- `totokens()` – tokenizes the cleaned sentences into word lists (tokens) for downstream vectorization or language-model tasks.

4. **Persist the Cleaned Data as a Binary File**

```
with open(f"Data\\cleaned_data\\{TYPE}_cleaned.pkl", "wb") as f:
    pickle.dump(data, f)
```

Serializes the cleaned data object to disk in binary format with `pickle`, enabling rapid reuse for model training or further analysis.

5. **Run the Analysis in Batch for All MBTI Types**

```
for T in tqdm(MBTI_types):
    analyze_data_p1(T)
```

Iterates over `MBTI_types` (assumed to include all 16 personality types) and executes `analyze_data_p1()` for each one, enabling efficient, consistent batch processing across the entire dataset.

6.1.2 LDA modeling

The cleaned data are organized by MBTI types, each as a tokenized and preprocessed text collection stored in a structured `pd.DataFrame`. To perform LDA modeling, the input must be converted into a **Bag-of-Words (BoW) corpus**, where each document is represented as a list of (word_id, frequency) tuples. This requires mapping each unique word to a distinct integer ID, resulting in a **dictionary** that captures the vocabulary of the entire corpus. In this assignment, we apply filtering by removing words that appear in more than 20% of documents (`no_above=0.2`) and those that appear in fewer than 50 documents (`no_below=50`) to retain only representative terms.

```
import gensim.corpora as corpora
from gensim.models import CoherenceModel
def constract_initial_dict(source,no_above,no_below):

    output = {T: {
        "original_text": [],
    } for T in MBTI_types}

    output["all_original_text"]=[]
    for T in tqdm(MBTI_types):
        for i in source[T].data.index:
            temp=source[T].data.loc[i,"posts"]
            output[T]["original_text"].append(temp)
        output["all_original_text"].extend(output[T]["original_text"])
    output["overall_dict"]=corpora.Dictionary(output["all_original_text"])
    output["overall_dict"].filter_extremes(no_above=no_above,no_below=
      no_below)
    output["overall_dict"].compactify()
    print("Size of dictionary:",len(output["overall_dict"]))
    output["all_corpus"]=[output["overall_dict"].doc2bow(post_token) for
      post_token in output["all_original_text"]]
    return output
initial_dict=constract_initial_dict(source=cleaned_data,
                                    no_above=0.20,
                                    no_below=50)
with open("Data/initial_dict.pkl",'wb') as f:
    pickle.dump(initial_dict,f)
```

For a convenient inspection, we computed the overall term-frequency distribution by summing each token's bag-of-words counts across the entire corpus and stored the result table as a CSV file:

```
def check_corpus(corpus,dict,name=''):
    result=pd.DataFrame(
        [
            list(range(len(dict))),
            [0]*len(dict)
        ]
    ).T
    result.columns=["word","frequency"]
```

```
    for post in tqdm(corpus):
        for word_tuple in post:
            result.loc[word_tuple[0],"frequency"]+=word_tuple[1]
    for i in result.index:
        result.loc[i,"word"]=dict[i]
    result=result.sort_values(by="frequency",ascending=False)
    result.to_csv(f"Data/{name}id2word_result.csv")
```

Selecting an appropriate number of topics is crucial for LDA modeling. If the topic number is too small, unrelated content may be merged into the same topic, reducing modelinterpretability. Conversely, an excessively large number of topics may fragment coherent semantic themes, leading to redundancy and overlap among topics. To determine the optimal number of topics for the LDA model, we will first train several temporary LDA models (`temp_lda_model`) with different numbers of topics. The semantic coherence and interpretability of the resulting topics were evaluated using the c_v coherence score, which measures how consistently related the top words within each topic are.

```
with open("Data/initial_dict.pkl","rb") as f:
    initial_dict=pickle.load(f)
import gensim.corpora as corpora
import gensim
from gensim.models import LdaMulticore,CoherenceModel

def optimize_topic_num(
  start,
  end,
  step,
  dict=initial_dict["overall_dict"],
  corpus=initial_dict["all_corpus"],
  text=initial_dict["all_original_text"]
):
  output=pd.Series({},dtype=float)
  topic_num_range=range(start, end+1, step)
  for topic_num in tqdm(topic_num_range, desc="Calculating optimal topic
    number"):
    # Train the LDA model (on all post data)
    temp_lda_model=LdaMulticore(
        corpus=corpus,    # Use the bag-of-words corpus of all posts
        id2word=dict,     # Use the global dictionary
```

```
            num_topics=topic_num,
            random_state=100,
            chunksize=100,    # Reduce chunksize to speed up update frequency
            passes=10,        # Reduce passes to shorten total training time
            iterations=50,    # Specify the number of iterations per pass
            alpha=0.01,       # Use a smaller fixed value to promote topic sparsity
            eta=0.01,         # Use a smaller fixed value to promote word sparsity
            per_word_topics=False,
            workers=None
        )

        # Evaluate the model
        temp_chmodel=CoherenceModel(
            model=temp_lda_model,
            texts=text,
            dictionary=dict,
            coherence="c_v"
        )
        output[topic_num]=temp_chmodel.get_coherence()
    print(output)
```

After a few trials, we find that 19 topics gives us the highest c_v coherence score.

```
计算全局模型主题数:    0%|
19      0.524685
20      0.488212
21      0.467017
22      0.472131
23      0.482915
24      0.470117
dtype: float64
```

Figure 6.1: Coherence scores for topic numbers 19 to 24. The highest score is observed at 19 topics, suggesting it as the optimal choice for this model.

We train the final LDA model (`lda_model`) with 19 topics using optimized settings, evaluate the model with c_v coherence score and save the model outputs, cleaned corpus, and topic descriptions for further analysis.

```
# Train the optimized final model with enhanced parameters for better convergence and topic separation
topics = 19

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
# Optimized parameters for better convergence and topic distinction
lda_model = LdaMulticore(
    corpus=initial_dict["all_corpus"],
    id2word=initial_dict["overall_dict"],
    num_topics=topics,
    random_state=100,
    chunksize=100,            # Significantly reduced: increases update frequency and improves convergence
    passes=300,               # Moderately reduced: adjusted to work with other optimized parameters
    iterations=150,           # Newly added: increase iterations per pass
    alpha=0.01,               # Changed from 'asymmetric' to a small value: promotes document-topic sparsity and improves topic distinction
    eta=0.01,                 # Changed from 'auto' to a small value: promotes word-topic sparsity and reduces topic mixing
    decay=0.5,                # Newly added: controls learning rate decay, improving convergence stability
    offset=1.0,               # Newly added: initial value for learning rate
    minimum_probability=0.01, # Newly added: filters out low-probability topic assignments
    per_word_topics=False,
    workers=None,             # Enables parallelization to speed up training
    eval_every=20             # Reduces evaluation frequency to lower computational cost
)
# Model evaluation
chmodel = CoherenceModel(
        model=lda_model,
        texts=initial_dict["all_original_text"],
        dictionary=initial_dict["overall_dict"],
        coherence="c_v"
    )
```

```
cv=chmodel.get_coherence()
cv
# Create unique directories for each LDA model
# That's because all variables are unique for each LDA model due to
  different stopword set
model_id=f"{topics}_{str(cv)[2:6]}"
# Model ID are designed as "[number of topics]_[CV score]", is unique for
  each model

path=f"output/lda_model/lda_{model_id}"
if not os.path.exists(path):
    os.makedirs(path)

path=f"output/lda_model/lda_{model_id}/cleaned_data"
if not os.path.exists(path):
    os.makedirs(path)

path=f"output/lda_model/lda_{model_id}/visualization"
if not os.path.exists(path):
    os.makedirs(path)

# Save LDA model
with open(f"output/lda_model/lda_{model_id}/lda_{model_id}.pkl",'wb') as f:
    pickle.dump(lda_model,f)

# Save cleaned data
with open(f"output/lda_model/lda_{model_id}/cleaned_data/cleaned_data.pkl",
   "wb") as f:
    pickle.dump(cleaned_data,f)

# Save original text
with open(f"output/lda_model/lda_{model_id}/all_original_text.pkl","wb") as
   f:
        pickle.dump(initial_dict["all_original_text"],f)

# Get all topics words and weights
all_topics_words = lda_model.show_topics(num_topics=-1, num_words=40,
  formatted=False)

```

```python
markdown_content=f"## {topics} topics, cv={str(cv)[2:6]}\n\n"

for topic_id, topic_words_with_weights in all_topics_words:
        markdown_content += f"### Topic {topic_id}:\n"


        for word, weight in topic_words_with_weights:
            markdown_content += f"- `{word}`: {weight:.4f}\n"
        markdown_content += "\n"

# Save topic words and weights to markdown file
with open(f"output/lda_model/lda_{model_id}/lda_{model_id}.md", "w",
  encoding="utf-8") as f:
        f.write(markdown_content)
```

After examining the model outputs, we observed that some meaningless or unrepresentative words still remain in the model. This is primarily because the Twitter dataset contains a substantial amount of internet slang (e.g., ”omg”) that was not fully removed during the initial data cleaning process. To address this issue, we manually add these terms to the stopword list and re-run the LDA modeling process iteratively until we obtain a model with a high c_v coherence score and minimal noise from irrelevant terms.

The model we finally settled on is model `lda_19_5687`, with the highest high c_v coherence score 0.5687. With this model, we will visualize the features of MBTI Twitter topics.

6.1.3 Result Visualization and Topic Evaluation

To start up the topic evaluation process, we need to import necessary libraries; define MBTI types,dimensions and groups; as well as load LDA model and cleaned data.

```python
# Import necessary libraries
%matplotlib widget
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
import numpy as np
import pickle
```

```python
from tqdm.auto import tqdm
import gensim.corpora as corpora
from gensim.models import LdaModel
from collections import defaultdict
from data_clean import Data_to_Clean,Data_to_Analyze
import warnings
warnings.filterwarnings('ignore')
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Ensure the plots are displayed correctly
plt.rcParams['axes.unicode_minus']=False
sns.set_style("whitegrid")
sns.set_palette("husl")

# Define MBTI types
MBTI_types=[
    'istj','isfj','infj','intj',
    'istp','isfp','infp','intp',
    'estp','esfp','enfp','entp',
    'estj','esfj','enfj','entj'
]

# Define MBTI dimensions
mbti_dimensions={
    'E': ['estp','esfp','enfp','entp','estj','esfj','enfj','entj'],
    'I': ['istj','isfj','infj','intj','istp','isfp','infp','intp'],
    'S': ['istj','isfj','istp','isfp','estp','esfp','estj','esfj'],
    'N': ['infj','intj','infp','intp','enfp','entp','enfj','entj'],
    'T': ['intj','intp','entj','entp','istj','istp','estj','estp'],
    'F': ['isfj','infj','isfp','infp','esfj','enfj','esfp','enfp'],
    'J': ['istj','isfj','infj','intj','estj','esfj','enfj','entj'],
    'P': ['istp','isfp','infp','intp','estp','esfp','enfp','entp']
}

# Define MBTI groups
mbti_groups={
    "analysts":["intj","intp","entj","entp"],
```

```
    "diplomats":["infj","infp","enfj","enfp"],
    "sentinels":["istj","isfj","istp","isfp"],
    "explorers":["isfp","istp","estp","esfp"]
}

# Load LDA model and data
# Model ID are designed as "[number of topics]_[CV score]",is unique for each model


def load_lda_data():
    # Load LDA model
    lda_model=pickle.load(open(f"output/lda_model/lda_{model_id}/lda_{model_id}.pkl","rb"))
    print(f"Successfully loaded LDA model with {lda_model.num_topics} topics")

    # Load original text data
    all_original_text=pickle.load(open(f"output/lda_model/lda_{model_id}/all_original_text.pkl","rb"))
    print(f"Successfully loaded original text data with {len(all_original_text)} documents")

    return lda_model,all_original_text


# Load cleaned data grouped by MBTI types
def load_mbti_data():
    file_path=f"output/lda_model/lda_{model_id}/cleaned_data/cleaned_data.pkl"
    with open(file_path,'rb') as f:
        cleaned_data=pickle.load(f)
    print(f"Cleaned data loaded successfully")

    return cleaned_data

# Execute file loading
lda_model,all_original_text=load_lda_data()
mbti_cleaned_data=load_mbti_data()
```

To better interpret topics and detect patterns in the data, we construct a class `LDATopicAnalyzer`, which integrates the LDA model with the MBTI-annotated dataset. This class performs several key functions:

1. Transforms input texts into bag-of-words representations, computes topic distributions for each document, and groups these distributions by MBTI type to observe group-level thematic tendencies.

2. Supports generating interactive visualizations through pyLDAvis, which is created and saved to html.

3. Provides tools to extract representative keywords per topic and optionally identify and exclude noise topics that carry little interpretive value.

```python
# Create a class for LDA visualization
class LDATopicAnalyzer:
    def __init__(self,lda_model,texts,mbti_data):
        self.model=lda_model
        self.texts=texts
        self.mbti_data=mbti_data
        self.dictionary=lda_model.id2word
        self.corpus=[self.dictionary.doc2bow(text) for text in texts]
        self.noise_topic=[]
        self.topic_distributions=None
        self.mbti_topic_distributions=None

    def create_pyldavis_visualization(self,save_path=f"final_output/lda_visualization.html"):

        print("Creating pyLDAvis visualization...")
        # Prepare pyLDAvis visualization
        vis_data=gensimvis.prepare(
            self.model,
            self.corpus,
            self.dictionary,
            sort_topics=False
        )

        # Save as HTML file
```

```
        pyLDAvis.save_html(vis_data,save_path)
        print(f"pyLDAvis visualization saved to: {save_path}")

        # Display in notebook
        pyLDAvis.enable_notebook()
        return pyLDAvis.display(vis_data)

    def get_topic_distributions(self):
        """Get topic distributions for documents"""
        print("Calculating topic distributions...")

        topic_distributions=[]
        for doc_bow in tqdm(self.corpus,desc="Calculating topic distributions"):
            doc_topics=self.model.get_document_topics(doc_bow,minimum_probability=0)
            topic_probs=[prob for _,prob in doc_topics]
            topic_distributions.append(topic_probs)

        self.topic_distributions=np.array(topic_distributions)
        return self.topic_distributions

    def calculate_mbti_topic_distributions(self):
        """Calculate topic distributions for each MBTI type"""
        print("Calculating topic distributions for each MBTI type...")

        mbti_topic_dist={}

        for mbti_type in MBTI_types:
            if mbti_type in self.mbti_data and len(self.mbti_data[mbti_type].data) > 0:
                # Create corpus for documents of this MBTI type
                mbti_corpus=[self.dictionary.doc2bow(doc) for doc in self.mbti_data[mbti_type].data["posts"]]

                # Calculate topic distributions
                topic_sums=np.zeros(self.model.num_topics)
                doc_count=0

```

```
                for doc_bow in mbti_corpus:
                    doc_topics=self.model.get_document_topics(doc_bow,
                      minimum_probability=0)
                    for topic_id,prob in doc_topics:
                        topic_sums[topic_id]+=prob
                    doc_count+=1

                # Calculate average topic distributions
                if doc_count>0:
                    mbti_topic_dist[mbti_type]=topic_sums/doc_count
                else:
                    mbti_topic_dist[mbti_type]=np.zeros(self.model.
                      num_topics)
            else:
                mbti_topic_dist[mbti_type]=np.zeros(self.model.num_topics)

        self.mbti_topic_distributions=mbti_topic_dist
        return mbti_topic_dist

    def get_topic_words(self,num_words=10):
        """Get keywords for each topic"""
        topic_words={}

        for topic_id in range(self.model.num_topics):
            words=self.model.show_topic(topic_id,topn=num_words)
            topic_words[topic_id]=[word for word,_ in words]
        return topic_words

    def add_noise_topics(self,*topic_ids):
        """Define noise topics"""
        for i in topic_ids:
            self.noise_topic.append(i)
# Create analyzer instance

analyzer=LDATopicAnalyzer(lda_model,all_original_text,mbti_cleaned_data)
print("LDA topic analyzer created successfully!")

# Create pyLDAvis interactive visualization
vis=analyzer.create_pyldavis_visualization()
```

```
vis
```

The interactive topic visualization generated by pyLDAvis can be accessed at the following link: https://dominicmin.github.io/Intro_to_DS_Assignment/lda_visualization.html. fig. 6.2 shows the initial state of the HTML file:

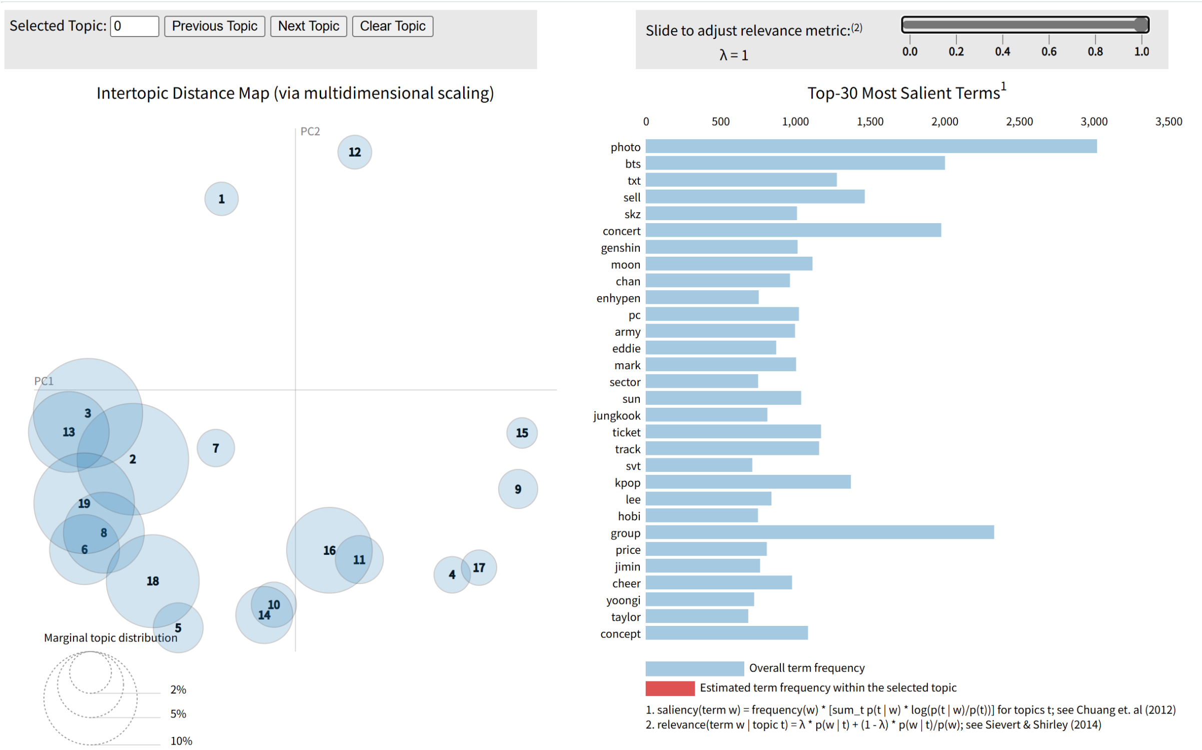


Figure 6.2: Screenshot of the initial state of the HTML file

Note: The topic indices in the HTML file are offset by +1 compared to the LDA model output (i.e., Topic 1 in HTML is Topic 0 in LDA).

As shown in fig. 6.2, on the left side of the HTML file, the **Intertopic Distance Map** visualizes the relationships between topics in two dimensions. A possible interpretation is that each bubble represents a distinct topic, with its area indicating the proportion of tokens (words) in the entire corpus attributed to that topic. Larger bubbles correspond to topics that account for a greater share of the corpus. The position and overlapping of each bubble reflects semantic similarity—the farther apart two bubbles are and the less the bubbles overlap, the less semantically related the corresponding topics. However, as will be demonstrated later, this interpretation fails to accurately characterize certain topics.

On the right side of the HTML page, the adjustable relevance metric of the selected topic is presented alongside the Top-30 Most Salient Terms. For each term, its overall frequency in the corpus is displayed as a blue histogram, while its estimated frequency within the selected topic is shown in red. Users can adjust the value of λ to observe changes in term saliency and the relationship between a term's overall frequency and its topic-specific relevance.

When λ is slided to the position 0, the terms that are unique in this topic are prior exhibited. These words are characterized by high distinctiveness, meaning they appear much more frequently in the selected topic compared to their frequency across the entire corpus. As a result, they serve as strong discriminators, helping to differentiate this topic from others. Although such terms may not be the most frequent within the topic itself, they often carry greater semantic specificity and are particularly useful for interpreting nuanced or domain-specific themes. However, because this setting emphasizes uniqueness over prevalence, it may occasionally highlight low-frequency or noisy terms, which should be interpreted with caution in topic labeling.

Meanwhile, when λ is slided to the position 1, these words represent the most commonly occurring terms in the topic and therefore reflect its core semantic content. This setting is particularly useful for understanding the dominant themes or main discourse of the topic. However, because it does not account for how exclusive a term is to the topic, some high-frequency terms that are also common in other topics may be included, potentially reducing the distinctiveness of the topic's representation.

From fig. 6.2, we can gain an overview of our modeling outcome. Topic 18 has the largest bubble, indicating that it is the most prevalent topic in the corpus. Topics 9, 15, 4, and 17 are located farther away from the other topics, suggesting a clear semantic distinction. In contrast, Topics 3, 13, 2, 8, 6, 19, and 18 exhibit notable spatial overlap, indicating only minor differences in their semantic content. In the whole corpus, the most reoccurring words are mostly related to the entertainment fandom and game fandom.

By selecting different topics and adjusting the value of λ, we can summarize the central theme of each topic, determine whether it is a noise topic, and assess its degree of semantic coherence.

Take topic 12 as an example, when λ is set to 0, the most relevant terms—such as trump, biden, republican, democrat, congress, and abortion—are highly distinctive and strongly associated with U.S. political discourse. These terms are not only topically specific but also exclusive to this topic, suggesting a focused and meaningful theme centered around American politics, government institutions, and social issues. In contrast, when λ is increased to 1, the top terms—such as state, country, child, gun, and law—shift toward higher-frequency words within the topic. Although some of these terms are more general, they still retain political relevance and semantic consistency with the topic's core, indicating that the theme is robust across different relevance metrics. In conclusion, the presence of consistently interpretable and contextually appropriate terms at both extremes of the λ scale demonstrates that topic 13 exhibits a relatively high degree of semantic coherence. The lack of function words, formatting artifacts, or off-topic vocabulary suggests that this is not a noise topic, but rather a well-defined and meaningful cluster within the corpus.

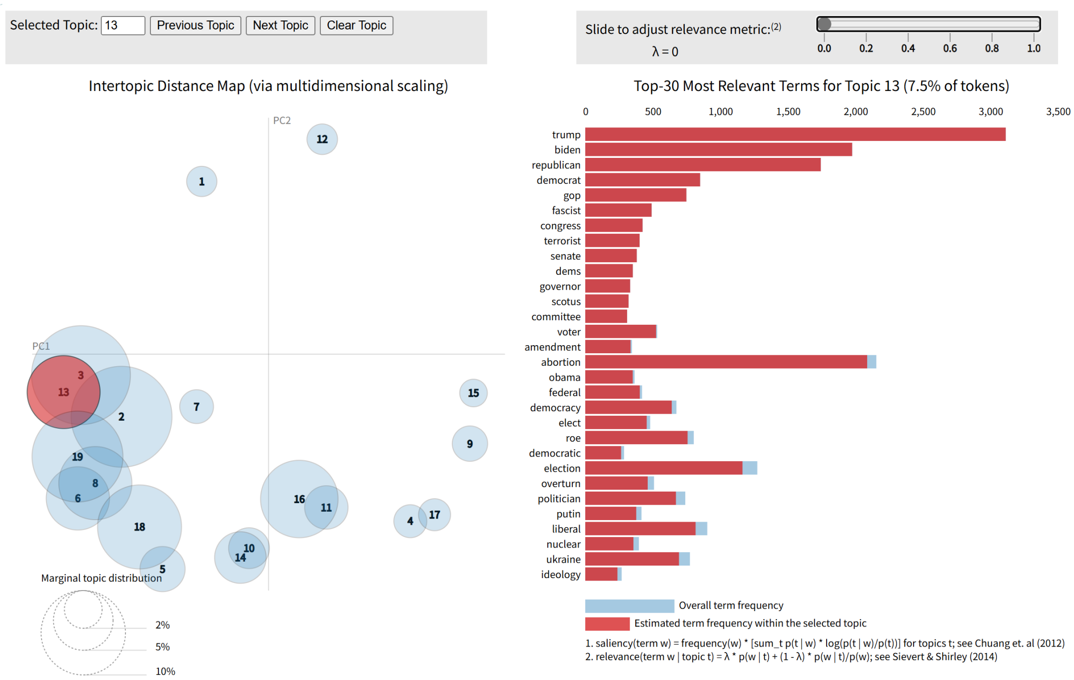


Figure 6.3: HTML page with Topic 12 selected and λ=0

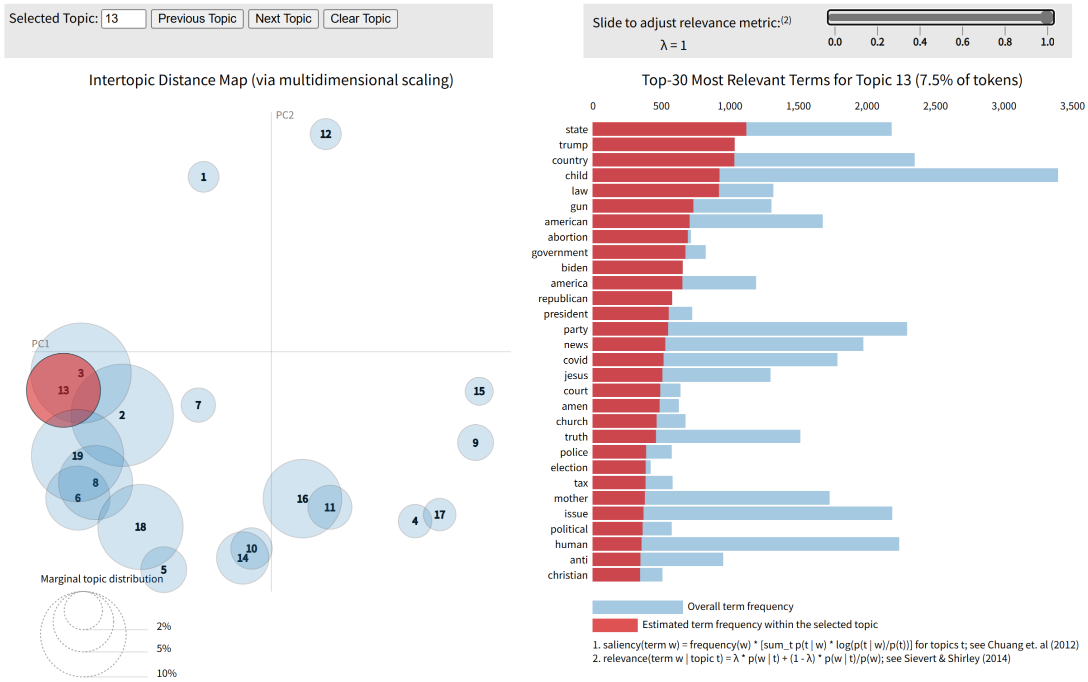


Figure 6.4: HTML page with Topic 12 selected and λ=0

By applying this method to all the topics, we can conclude the following topic modeling result:

- High semantic coherence - 8 topics:

  Topic 0: Astrology/Zodiac

  Topic 3: K-pop Groups (TXT/NCT)

  Topic 4: Western TV Shows (Stranger Things/Heartstopper)

  Topic 6: Gaming (Genshin Impact)

  Topic 8: K-pop Groups (SEVENTEEN/ATEEZ)

  Topic 9: Western Pop Music/Celebrities

  Topic 10: K-pop Group (BTS)

  Topic 14: K-pop Group (ENHYPEN)

  Topic 16: K-pop Group (Stray Kids)

- Medium semantic coherence - 6 topics:

  Topic 2: Academic/Business/Personal Development

  Topic 5: Gender Identity/LGBTQ+/Mental Health

Topic 7: Food/Weight Management/Eating Disorders

Topic 11: Merchandise Trading/Collectibles

Topic 12: Politics/Social Issues

Topic 17: Anime/Manga/Fan Culture

Topic 1: Daily Life and traveling

- Low semantic coherence - 1 topic:

  Topic 15: K-pop Industry General

- Noise Topics - 2 topics:

  Topic 13: Mixed Social Media Expressions

  Topic 18: Generic Terms/Mixed Content

Interestingly, when linking the interpreted topics to their corresponding bubbles, we observe that although Topic 2 (Academic/Business/Personal Development) and Topic 12 (Politics/Social Issues) show considerable overlap in the Intertopic Distance Map, they are not semantically similar. This mismatch occurs because high-dimensional topic data is projected onto a two-dimensional space.

6.1.4 Topic Distribution

We begin by calculating and visualizing topic distribution.

```
# Calculate topic distributions for MBTI types

mbti_topic_dist=analyzer.calculate_mbti_topic_distributions()
topic_words=analyzer.get_topic_words()

# Create topic distribution DataFrame
topic_dist_df=pd.DataFrame(mbti_topic_dist).T
topic_dist_df.columns=[f"Topic {i}" for i in range(len(topic_dist_df.
  columns))]
topic_dist_df.drop([f"Topic {i}" for i in analyzer.noise_topic],
                   axis=1,
                   inplace=True)

```

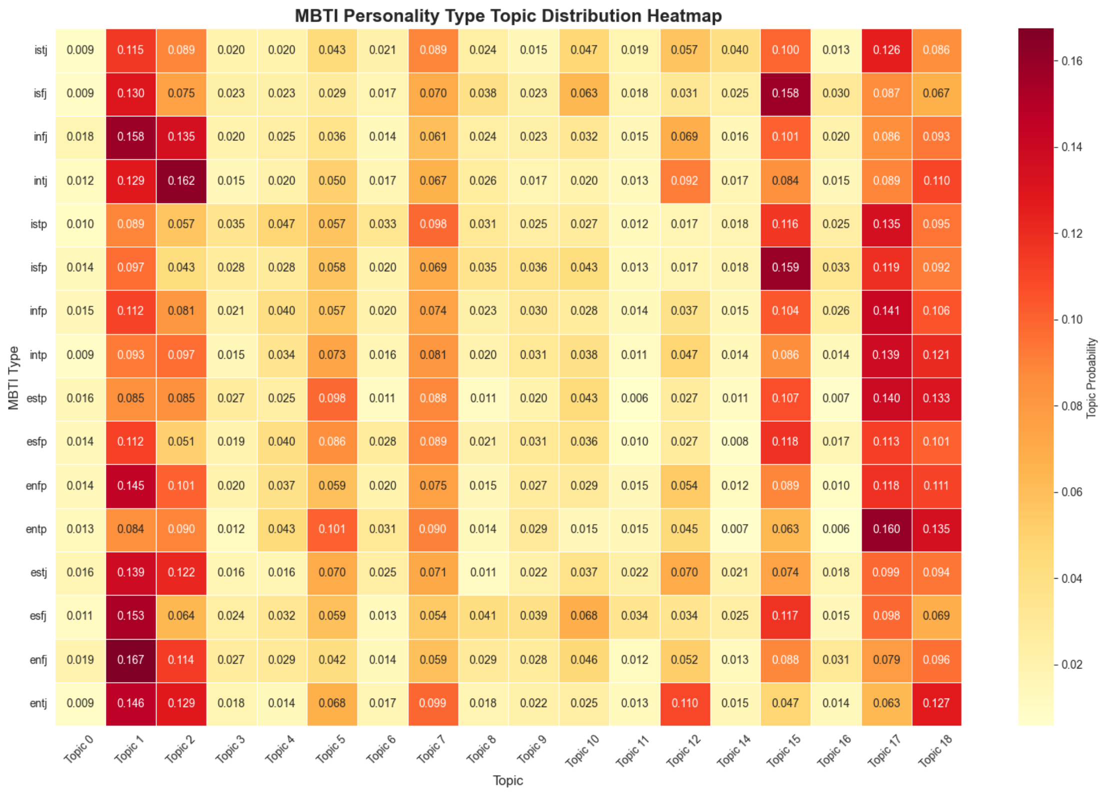
```
print("MBTI type topic distribution calculation completed!")
print(f"Topic distribution matrix shape: {topic_dist_df.shape}")

# Display first few rows
print("\nPreview of topic distributions for each MBTI type:")
display(topic_dist_df.head())
```

```
Calculating topic distributions for each MBTI type...
MBTI type topic distribution calculation completed!
Topic distribution matrix shape: (16, 18)

Preview of topic distributions for each MBTI type:
```

	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9	Topic 10	Topic 11	Topic 12	Topic 14	Topic 15
istj	0.008851	0.115248	0.088992	0.020304	0.020248	0.043244	0.021340	0.088940	0.024473	0.015487	0.046689	0.018558	0.057390	0.040083	0.099723
isfj	0.009061	0.129999	0.074834	0.022828	0.023307	0.028510	0.017359	0.069671	0.038191	0.022679	0.063308	0.017913	0.031093	0.024531	0.157857
infj	0.017712	0.158324	0.134677	0.020251	0.025003	0.035576	0.014002	0.060592	0.023622	0.022519	0.031834	0.014991	0.068709	0.015630	0.100966
intj	0.011780	0.128567	0.161768	0.014841	0.020404	0.050019	0.016520	0.066708	0.026187	0.017296	0.020154	0.012905	0.092029	0.017059	0.084242
istp	0.009948	0.088690	0.056644	0.034672	0.046965	0.057443	0.033386	0.098363	0.030848	0.024917	0.027006	0.012163	0.017179	0.018405	0.116180

Figure 6.5: Preview of first few rows

A heatmap is generated based on the computed values to visually represent the results.



The heatmap provides an intuitive visual indication of topic popularity. Specifically, Topics 1 (Daily Life and traveling), 2 (Academic/Business/Personal Development), 15 (K-pop Industry General), and 17 (Anime/Manga/Fan Culture) exhibit darker red shades, suggesting they are more frequently discussed. A more rigorous, quantitative analysis of topic popularity will be presented further on.

To assist with our interpretation with the heatmap, we calculate the overall popularity of each topic.

```python
# Create topic keyword cloud summary
def create_topic_wordcloud_summary(topic_words,topic_dist_df):
    """Create topic keyword summary"""
    print("=" * 60)
    print("Topic keyword summary")
    print("=" * 60)

    # Calculate overall popularity of each topic
    topic_popularity=topic_dist_df.mean().sort_values(ascending=False)

    for i,(topic_idx,popularity) in enumerate(topic_popularity.items()):
        topic_num=int(topic_idx.replace('Topic',''))
        print(f"\nTopic {topic_num} (Popularity: {popularity:.3f}):")
        print(f"Keywords: {','.join(topic_words[topic_num][:8])}")

        # Find the MBTI type that most prefers this topic
        topic_col=f"Topic {topic_num}"
        if topic_col in topic_dist_df.columns:
            top_mbti=topic_dist_df[topic_col].nlargest(3)
            print(f"Most preferred MBTI types: {','.join([f'{mbti}({score
              :.3f})' for mbti,score in top_mbti.items()])}")

        if i >= 9:  # Only show top 10 topics
            break


create_topic_wordcloud_summary(topic_words,topic_dist_df)
```

This gives us the following summary:

```
============================================================
Topic keyword summary
============================================================

Topic 1 (Popularity: 0.122):
Keywords: photo,coffee,covid,trip,college,husband,felt,child
Most preferred MBTI types: enfj(0.167),infj(0.158),esfj(0.153)

Topic 17 (Popularity: 0.112):
Keywords: anime,chapter,fic,manga,holy,commission,au,arc
Most preferred MBTI types: entp(0.160),infp(0.141),estp(0.140)

Topic 18 (Popularity: 0.102):
Keywords: black,beat,star,war,holy,animal,ball,single
Most preferred MBTI types: entp(0.135),estp(0.133),entj(0.127)

Topic 15 (Popularity: 0.101):
Keywords: group,member,comeback,kpop,debut,dance,stalker,concert
Most preferred MBTI types: isfp(0.159),isfj(0.158),esfp(0.118)

Topic 2 (Popularity: 0.093):
Keywords: experience,self,human,study,thread,important,business,language
Most preferred MBTI types: intj(0.162),infj(0.135),entj(0.129)

Topic 7 (Popularity: 0.077):
Keywords: fast,weight,milk,fat,cream,ice,coffee,smell
Most preferred MBTI types: entj(0.099),istp(0.098),entp(0.090)

Topic 5 (Popularity: 0.061):
Keywords: trans,sex,gender,act,male,black,autistic,dick
Most preferred MBTI types: entp(0.101),estp(0.098),esfp(0.086)

Topic 12 (Popularity: 0.049):
Keywords: state,trump,country,child,law,gun,american,abortion
Most preferred MBTI types: entj(0.110),intj(0.092),estj(0.070)

Topic 10 (Popularity: 0.037):
Keywords: bts,army,jungkook,jimin,hobi,yoongi,taehyung,tae
Most preferred MBTI types: esfj(0.068),isfj(0.063),istj(0.047)

Topic 4 (Popularity: 0.030):
Keywords: eddie,steve,mike,stranger,robin,max,strange,el
Most preferred MBTI types: istp(0.047),entp(0.043),infp(0.040)
```

With the aid of the summary statistics, we can find distinct MBTI engagement patterns across different semantic domains:

- Topic 15 (K-pop fandom activities): characterized by group debuts, comebacks, and concerts; demonstrates strong engagement among ISFP, ISFJ, and ESFP types, underscoring the intersection of sensory enjoyment and emotional engagement.

- Topic 14 (lightweight, emotionally-driven daily expression): resonates with SP types who prioritize spontaneity and immediate experiential sharing.

- Topic 7 (sensory experiences related to food and comfort): emphasizes fast food, coffee, and cream, aligning with ISTP and ESTP preferences for direct sensory satisfaction.

- Topic 17 (strategic, abstract, and fandom-oriented discussions): involves anime, manga, and fan fiction; attracts ENTP, INFP, and ESTP types, indicating alignment with analytical and imaginative discourse.

- Topic 2 (experiential and intellectual reflections): appeals to INTJ, ENTJ, and INFJ types, reflecting their preference for profound, thoughtful, and self-reflective content.

- Topic 18 (adventurous and competitive themes): includes sports and competitions; engages ENTP, ESTP, and ENTJ personalities drawn to intense experiences.

- Topic 5 (identity and socially charged discussions): covers gender, sexuality, and societal roles; garners interest from ENTP, ESTP, and ESFP types open to explorative and provocative topics.

- Topic 12 (politically charged discourse): focuses on societal debates like Trump, guns, and abortion; attracts ENTJ, INTJ, and ESTJ types inclined toward policy and governance.

- Topic 10 (BTS fandom): features BTS member names; resonates with ESFJ and ISFJ types, highlighting community-oriented engagement.

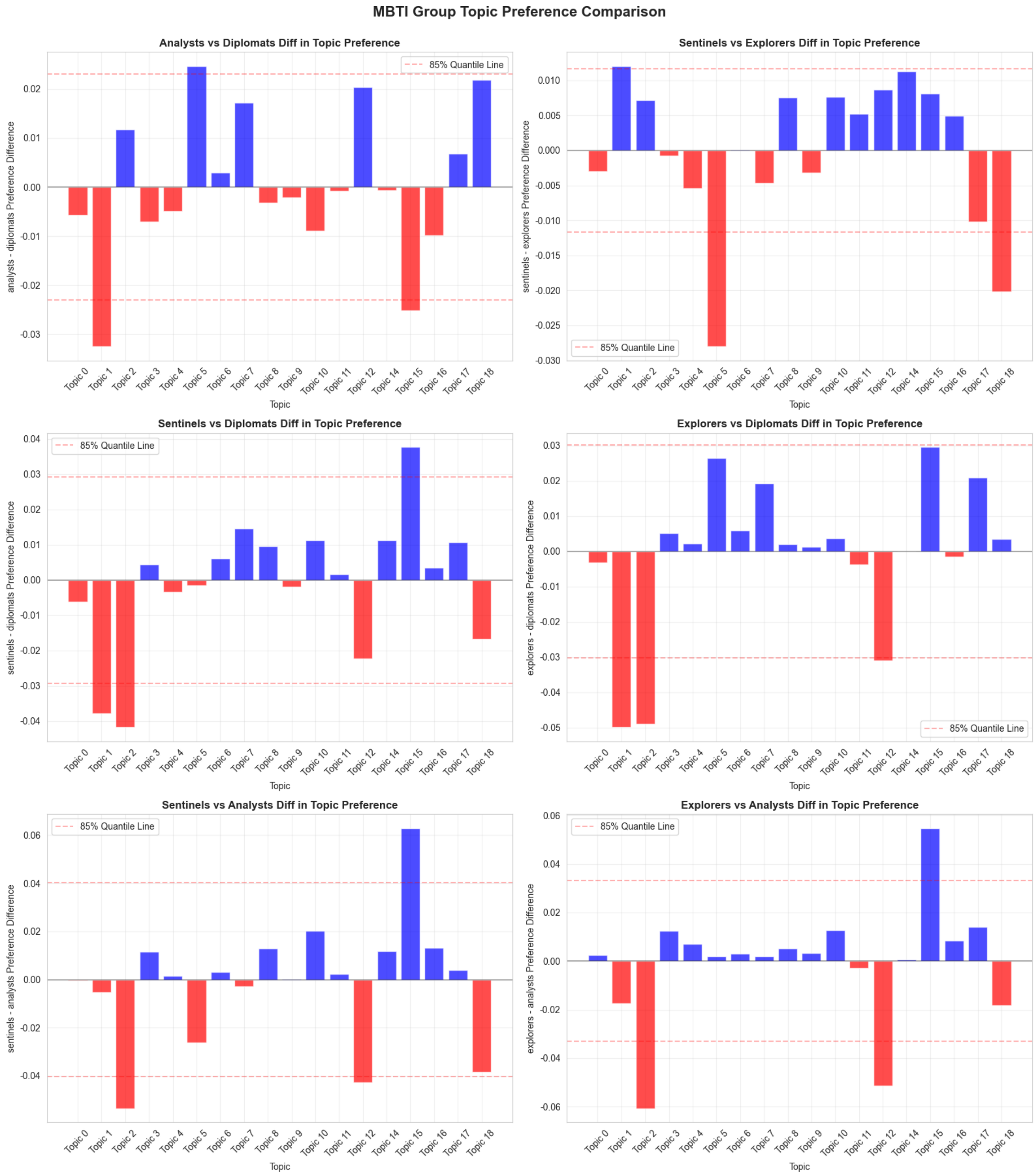
- Topic 4 (Stranger Things franchise): references characters like Eddie and Steve; shows engagement from ISTP, ENTP, and INFP types, indicating analytical and introspective affinities.

Having observed engagement patterns of individual MBTI types, we compare the topic preference patterns of different MBTI groups.

```python
# Create MBTI dimension topic preference analysis
def analyze_mbti_dimensions(topic_dist_df,mbti_dimensions=mbti_dimensions):
    """Analyze topic preferences for MBTI four dimensions"""
    dimension_analysis={}

    for dim,types in mbti_dimensions.items():
        # Calculate average topic distribution for this dimension type
        dim_types_in_data=[t for t in types if t in topic_dist_df.index] #
          t is a single type
```

```python
        dimension_analysis[dim]=topic_dist_df.loc[dim_types_in_data].mean()  # mean of the topic distribution for this dimension/group type

    return pd.DataFrame(dimension_analysis)

dimension_pairs=[('E','I'),('S','N'),('T','F'),('J','P')]
pair_names=['Extrovert vs Introvert','Sense vs Intuition','Thinking vs Feeling','Judging vs Perceiving']


def create_dimension_comparison(dimension_df,file_name,dimension_pairs=dimension_pairs,pair_names=pair_names,comment=None):
    """Create MBTI dimension topic preference comparison chart"""
    fig,axes=plt.subplots(int(len(dimension_pairs)/2),2,figsize=(16,6*int(len(dimension_pairs)/2)))
    axes=axes.flatten()

    for i,((dim1,dim2),pair_name) in enumerate(zip(dimension_pairs,pair_names)):
        if dim1 in dimension_df.columns and dim2 in dimension_df.columns:
            # Calculate difference
            diff=dimension_df[dim1]-dimension_df[dim2]

            # Create bar chart
            x_pos=range(len(diff))
            colors=['red' if x < 0 else 'blue' for x in diff]

            axes[i].bar(x_pos,diff,color=colors,alpha=0.7)
            axes[i].axhline(y=diff.apply(abs).quantile(q=0.85),color='red',linestyle='--',alpha=0.3,label='85% Quantile Line')
            axes[i].axhline(y=-diff.apply(abs).quantile(q=0.85),color='red',linestyle='--',alpha=0.3)
            axes[i].axhline(y=0,color='black',linestyle='-',alpha=0.3)
            axes[i].set_title(f'{pair_name} Diff in Topic Preference',fontsize=12,fontweight='bold')
            axes[i].set_xlabel('Topic')
            axes[i].set_ylabel(f'{dim1} - {dim2} Preference Difference')
            axes[i].set_xticks(x_pos)
            axes[i].set_xticklabels([j for j in list(topic_dist_df.columns)
```

```
             ],rotation=45)
            axes[i].grid(True,alpha=0.3)
            axes[i].legend()
    fig.suptitle("MBTI Group Topic Preference Comparison",fontsize=16,fontweight='bold',y=1)
    plt.tight_layout()
    plt.figtext(0.5, 0.02,comment,ha='center',fontsize=10)
    plt.savefig(f'output/lda_model/lda_{model_id}/visualization/{file_name}.png',dpi=300,bbox_inches='tight')
    plt.savefig(f'final_output/{file_name}_drop13_18.png',dpi=300,bbox_inches='tight')
    plt.show()


dimension_df=analyze_mbti_dimensions(topic_dist_df)
print("MBTI dimension topic preference analysis:")
display(dimension_df.head())

# create_dimension_comparison(dimension_df,"mbti_dimension_comparison")
group_pairs=[
            ('analysts','diplomats'),
            ('sentinels','explorers'),
            ('sentinels','diplomats'),
            ('explorers','diplomats'),
            ('sentinels','analysts'),
            ('explorers','analysts')
             ]
group_names=['Analysts vs Diplomats',
             'Sentinels vs Explorers',
             'Sentinels vs Diplomats',
             'Explorers vs Diplomats',
             'Sentinels vs Analysts',
             'Explorers vs Analysts'
             ]

group_df=analyze_mbti_dimensions(topic_dist_df,mbti_groups)
create_dimension_comparison(group_df,"mbti_group_comparison",group_pairs,group_names)
```



For a unified comparison, we set the standard of significantly different as any absolute topic-preference gap that surpasses the 85th-percentile threshold, which is indicated by the red dashed line in each subplot.

From these 6 comparisons we can come up with the following conclusions:

1. Analysts show a clear surplus of discussion in the “Gender / Mental-Health” and “Politics / Social Issues” topics, whereas Diplomats contribute markedly less to these threads. This suggests that the NT temperaments are more inclined toward idea-driven or policy-oriented debates, while the NF group devotes its energy to lighter, experience-based con-

versations such as travel logs and casual fandom chatter.

2. When Sentinels are compared with Explorers, the former post more frequently about day-to-day planning, travel itineraries, and the buying or trading of collectibles. Explorers, in contrast, engage less with these logistics-heavy themes and more with personal identity talk. This difference implies that SJ personalities favour order and tangible exchanges, while SP personalities gravitate toward spontaneous self-expression.

3. Sentinels dominate K-pop industry news and merchandise exchange, yet they underperform on topics related to academic or personal growth. Therefore, the data indicate a Sentinel preference for operational or execution-oriented content, whereas Diplomats continue to focus on introspection and self-development narratives.

4. Explorers outpace Diplomats in discussions about ENHYPEN fandom, the game Genshin Impact, and LGBTQ identity issues, but fall behind in travel diaries, study tips, and political conversations. This result reinforces the notion that SP types pursue sensory entertainment and identity exploration, whereas NF types lean toward reflective or civic-minded themes.

5. Sentinels tweet far more about mainstream idols such as ENHYPEN and BTS, while Analysts turn their attention to academically oriented or self-improvement threads. We conclude that SJ users are deeply involved in collective fan activities, whereas NT users remain focused on knowledge and analytical depth.

6. Explorers lead Analysts in anime and manga chatter, yet they post significantly less about academic and political issues. The evidence points to an SP preference for leisure-centred pop-culture content, contrasting with an NT preference for intellectually demanding discussions.

6.1.5 MBTI type Clustering

Using LDA modeling, we have identified the topic preference different of MBTI personality types. To which extent does the estimated or stereotyped topic preference of the official MBTI groups align with their realistic behaviors? We cluster MBTI types to find out.

Reducing high-dimensional topic distributions into a three-dimensional principal component space is necessary for K-means clustering to be preformed. So, PCA dimensionality reduction is preformed before we preform K-means clustering

```
def create_mbti_clustering_analysis_3d(n_clusters,topic_dist_df=
  topic_dist_df):

    # Standardize data
    scaler=StandardScaler()
    scaled_data=scaler.fit_transform(topic_dist_df)

    # 3D PCA dimensionality reduction
    pca=PCA(n_components=3)
    pca_3d_data=pca.fit_transform(scaled_data)

    # K-means clustering (on 3D data)
    kmeans=KMeans(n_clusters=n_clusters, random_state=42)
    clusters=kmeans.fit_predict(pca_3d_data)  # Note: clustering on 3D data
```

After clustering, we print PCA variance to see if PCA dimensionality reduction is effective.

```
# Print variance explained
print(f"3D PCA variance explained:")
print(f"PC1: {pca.explained_variance_ratio_[0]:.3f}")
print(f"PC2: {pca.explained_variance_ratio_[1]:.3f}")
print(f"PC3: {pca.explained_variance_ratio_[2]:.3f}")
print(f"Total: {pca.explained_variance_ratio_.sum():.3f}")
```

The results are:

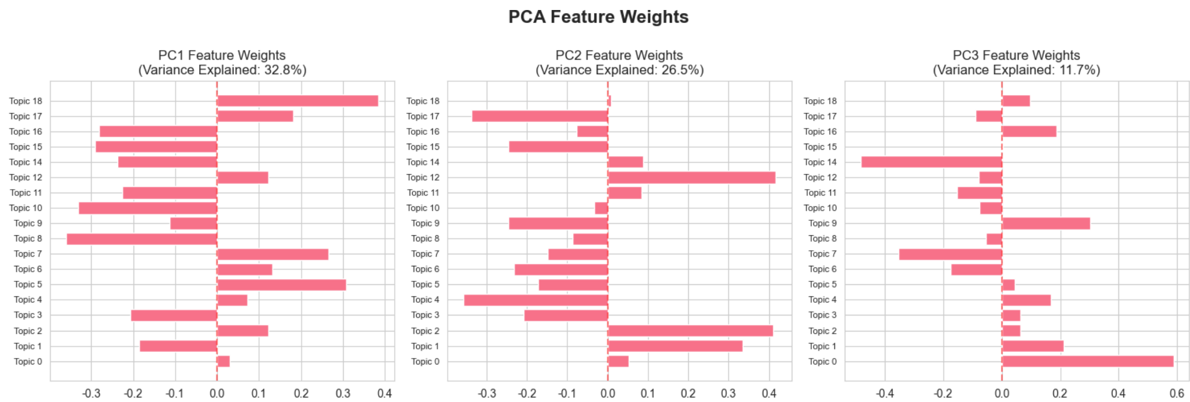
```
PC1: 0.328
PC2: 0.265
PC3: 0.117
Total: 0.709
```

With a total of 70.9% of information successfully reserved, our dimensionality reduction is successful.

To interpret our clustering results, it is necessary for us to find out what PC1, PC2, PC3 represents respectively. So we print out the main contributing features of the three dimensions respectively and plot out PCA feature weights.

```
PC1 Main contributing features (sorted by weight):
Topic 18    0.385
Topic 8     0.361
Topic 10    0.331
Topic 5     0.308
Topic 15    0.290
Name: PC1, dtype: float64

PC2 Main contributing features (sorted by weight):
Topic 12    0.417
Topic 2     0.411
Topic 4     0.358
Topic 17    0.338
Topic 1     0.334
Name: PC2, dtype: float64

PC3 Main contributing features (sorted by weight):
Topic 0     0.587
Topic 14    0.483
Topic 7     0.353
Topic 9     0.303
Topic 1     0.213
Name: PC3, dtype: float64
```

Figure 6.6: Printed Main contributing features



By analyzing the printed features and plotted results, we approximately conclude the meaning of each axis:

1. PC1: Leisure & Fandom vs. Everyday-life.

2. PC2: Politics–Rationality vs. Entertainment–Support.

3. PC3: Fragmented expressions vs. centralized fandom behavior

6 distinctive clusters can be distinguished from our clustering results.

```
# Print clustering results
print("\nClustering results:")
print("=" * 50)
for cluster_id in range(n_clusters):
```

```
        # Get mbti types in the current cluster
        mbti_in_cluster=[topic_dist_df.index[i] for i in range(len(clusters)) if clusters[i] == cluster_id]
        # Convert to uppercase and sort
        mbti_in_cluster_upper=sorted([mbti.upper() for mbti in mbti_in_cluster])
        print(f"Cluster {cluster_id + 1}: {', '.join(mbti_in_cluster_upper)}")
    print("=" * 50)
```

```
Clustering results:
==================================================
Cluster 1: ISFP, ISTP
Cluster 2: ENTJ, ESTJ, INTJ, ISTJ
Cluster 3: ESFJ, ISFJ
Cluster 4: ENFP, ENTP, ESFP, ESTP, INFP, INTP
Cluster 5: ENFJ, INFJ
==================================================
```

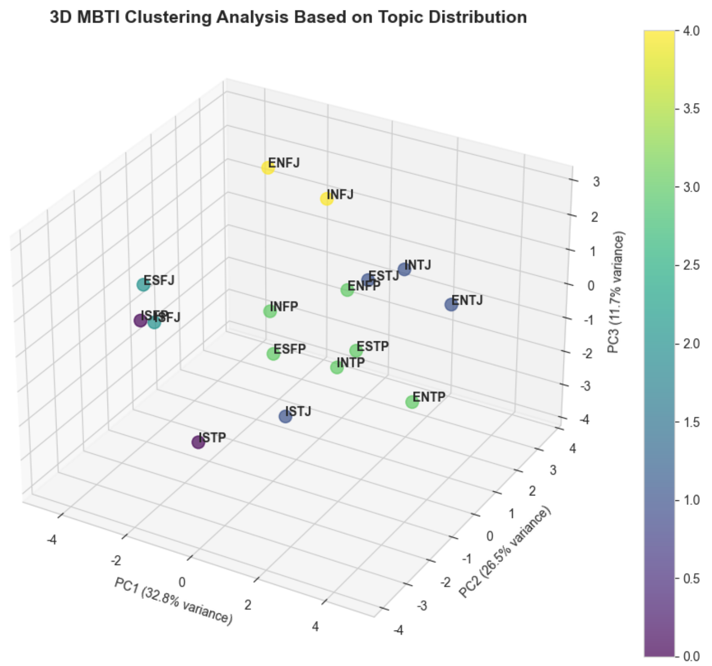
Figure 6.7: Printed clustering results

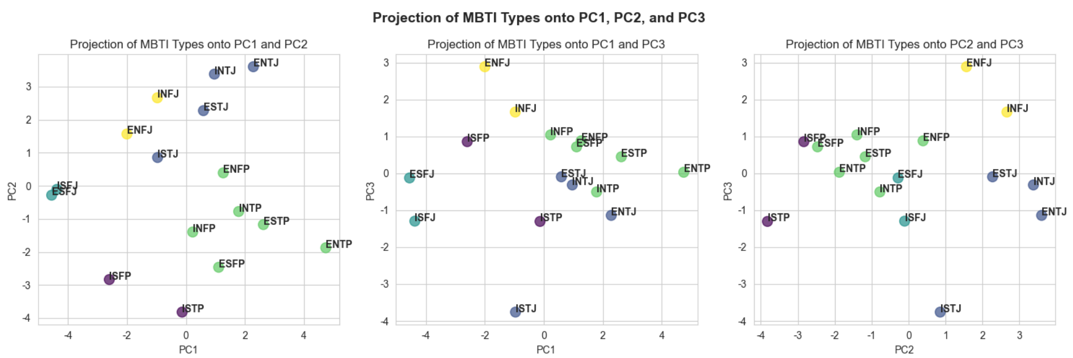
A 3-D scatter plot and it’s 2-dimensional projection is plotted to visualize the results.

```
# 3D visualization
fig=plt.figure(figsize=(12, 9))
ax=fig.add_subplot(111, projection='3d')

# 3D scatter plot
scatter=ax.scatter(pca_3d_data[:,0], pca_3d_data[:,1], pca_3d_data[:,2],
                    c=clusters, cmap='viridis', s=100, alpha=0.7)

# Add MBTI labels
for i, mbti_type in enumerate(topic_dist_df.index):
    ax.text(pca_3d_data[i,0], pca_3d_data[i,1], pca_3d_data[i,2],
           mbti_type.upper(), fontsize=10, fontweight='bold')

ax.set_xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.1%} variance)')
ax.set_ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.1%} variance)')
ax.set_zlabel(f'PC3 ({pca.explained_variance_ratio_[2]:.1%} variance)')

ax.set_title('3D MBTI Clustering Analysis Based on Topic Distribution',
            fontsize=14, fontweight='bold')
```

```python



    plt.colorbar(scatter)
    plt.show()


    return pca_3d_data, clusters
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))
    axes[0].scatter(cl_analysis[0][:,0], cl_analysis[0][:,1],
                            c=cl_analysis[1], cmap='viridis', s=100, alpha=0.7)
    axes[1].scatter(cl_analysis[0][:,0], cl_analysis[0][:,2],
                            c=cl_analysis[1], cmap='viridis', s=100, alpha=0.7)
    axes[2].scatter(cl_analysis[0][:,1], cl_analysis[0][:,2],
                            c=cl_analysis[1], cmap='viridis', s=100, alpha=0.7)
    axes[0].set_xlabel("PC1")
    axes[0].set_ylabel("PC2")
    axes[0].set_title("Projection of MBTI Types onto PC1 and PC2")
    axes[1].set_xlabel("PC1")
    axes[1].set_ylabel("PC3")
    axes[1].set_title("Projection of MBTI Types onto PC1 and PC3")
    axes[2].set_xlabel("PC2")
    axes[2].set_ylabel("PC3")
    axes[2].set_title("Projection of MBTI Types onto PC2 and PC3")
    for i, mbti_type in enumerate(topic_dist_df.index):
        axes[0].text(cl_analysis[0][i,0], cl_analysis[0][i,1],
                     mbti_type.upper(), fontsize=10, fontweight='bold')
        axes[1].text(cl_analysis[0][i,0], cl_analysis[0][i,2],
                     mbti_type.upper(), fontsize=10, fontweight='bold')
        axes[2].text(cl_analysis[0][i,1], cl_analysis[0][i,2],
                     mbti_type.upper(), fontsize=10, fontweight='bold')
    fig.suptitle("Projection of MBTI Types onto PC1, PC2, and PC3", fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.savefig(f"output/lda_model/lda_{model_id}/visualization/mbti_clustering_projection.png")
```

```
    plt.savefig(f"final_output/mbti_clustering_projection.png")
    plt.show()
```





Using the concluded meaning of each axis, we can come up with the following conclusions:

- Cluster 1: ISFP, ISTP — leisure-oriented, low political talk, fragmented style
- Cluster 2: ENTJ, ESTJ, INTJ, ISTJ — everyday & rational topics, highly centralised
- Cluster 3: ESFJ, ISFJ — life-focused, cooperative support behaviour
- Cluster 4: ENFP, ENTP, ESFP, ESTP, INFP, INTP — strong entertainment preference, fragmented expression

- Cluster 5: ENFJ, INFJ — politics / affect-driven, centralised fandom

This outcome aligns closely to the four MBTI groups, suggesting certain reliability of MBTI grouping.

6.2 Expressive Patterns of MBTI Personalities

6.2.1 Data Cleaning and Pre-processing

Expression pattern analysis shares the same initial steps as LDA modeling: removing URLs, emojis, and tags. Please refer to the LDA data-cleaning section for details. We only need to load the cleaned data for our expressive pattern analysis.

```
from data_clean import Data_to_Clean,Data_to_Analyze
import pickle
import os
import pandas as pd
import copy
import json
import logging
from tqdm.auto import tqdm
import matplotlib.pyplot as plt
tqdm.pandas()

MBTI_types=[
    'istj', 'isfj', 'infj', 'intj',
    'istp', 'isfp', 'infp', 'intp',
    'estp', 'esfp', 'enfp', 'entp',
    'estj', 'esfj', 'enfj', 'entj'
    ]
cleaned_data={T:None for T in MBTI_types}

for type in cleaned_data.keys():
    file_path=f"Data\\cleaned_data\\{type}_cleaned.pkl"
    try:
        with open(file_path, 'rb') as f:
            cleaned_data[type] = pickle.load(f)
    except FileNotFoundError:
        print(f"Error: File not found at {file_path}")
    except pickle.UnpicklingError:
```

```
            print(f"Error: Could not unpickle the file {file_path}. It might be corrupted or not a valid pickle file.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

To check whether we have successfully imported the cleaned dataset, we print out the 0th column, ”posts” row of infp.

```
infp=cleaned_data["infp"]
infp.data.loc[0,"posts"]
```

From the output, we can see that there is no URLs, emojis, and tags in our data, which means that we have successfully cleaned and read from the cleaned dataset. Our data is ready for analysis.

```
[50]:  ['forest',
         'elizabeth',
         'holly',
         'fuss',
         'seat',
         'pant',
         'best',
         'friend',
         'day',
         'yes',
         'decoration',
         'old',
         'day',
         'change',
         'present',
         'guess',
```

Figure 6.8: Cleaned Data printed out

6.2.2 Linguistic Style Patterns of MBTI Types

We select 4 distinct linguistic dimensions to represent linguistic styles:

1. Sentence Quantity: A larger sentence quantity suggests that the user tends to express themselves in shorter, more frequent statements. This may reflect a conversational, spontaneous communication style, often associated with extraversion or emotional openness.

2. Word Count: Higher word count may indicate a more elaborative or expressive communication habit, suggesting the user tends to provide more context or detail. This could reflect traits such as thoughtfulness, emotional depth, or even analytical thinking.

3. Upper-Case Ratio: A higher ratio of uppercase letters often signifies stronger emotional expression, emphasis, or a more dramatic tone. It may reveal higher emotional arousal or a more assertive communication style, which could be linked to personality traits like enthusiasm or dominance.

4. Reading Ease(reflected by both Flesch and GF index): Higher reading ease scores imply simpler sentence structures and more familiar vocabulary, often seen in casual or informal communication. In contrast, lower scores suggest complexity and abstractness, which could indicate intellectual engagement, formality, or introversion. In our project, we adopted two kinds of rating systems to evaluate the reading ease of a context. The higher the Flesch Reading Ease score is, the easier the context can be understood; the lower the GF index is, the easier the context can be understood.

These values are first computed before further analysis.

1. Class Definition & Constructor

```
class Data_to_Analyze(Data_to_Clean):
    def __init__(self, type, source=raw_data):
        super().__init__(source)
        self.data = self.data.loc[self.data["type"] == type].
          reset_index(drop=True)
        self.data_to_vec = None
        self.basic_identities = pd.Series({
            "type": type,
            # Number of sentences in a post
            "sentence_quantity": [],
            "ave_sentence_quantity": None,
            # Number of words in a post
            "word_count": [],
            "ave_word_count": None,
            # Ratio of upper case characters in a post
            "upper_ratio": [],
            "ave_upper_ratio": None,
            # Two readability indicators: Flesch Reading Ease and
              Gunning Fog Index
            "reading_ease": [],
            "ave_reading_ease": None,
            "GF_index": [],
```

```
            "ave_GF_index": None,
            # Overall sentiment indicator (VADER)
            "overall_vader_score": None
        })
```

Defines a class `Data_to_Analyze` that inherits from `Data_to_Clean`.

- **Data subset** – The constructor calls super().__init__(source) to initialize the parent class, then filters the dataset so that only rows whose type column equals the target MBTI type remain (self.data).
- **Feature container** – basic_identities is a pd.Series used to store a collection of text statistics: sentence count, word count, upper case ratio, readability metrics (Flesch Reading Ease and Gunning Fog Index) and an overall VADER sentiment score.

2. Sentence Count

```
def get_sentence_quantity(self):
    for post in self.data["posts"].values:
        self.basic_identities["sentence_quantity"].append(len(post))
    self.basic_identities["ave_sentence_quantity"] = ave(self.
      basic_identities["sentence_quantity"])
```

Iterates through every post (each post is already a list of sentences). The length of the list gives the number of sentences, which is appended to `sentence_quantity`. Finally, the helper `ave()` computes their mean.

3. Word Count

```
def get_word_count(self):
    for post in self.data["posts"].values:
        total = 0
        for sentence in post:
            total += len(sentence.split(" "))
        self.basic_identities["word_count"].append(total)
    self.basic_identities["ave_word_count"] = ave(self.
      basic_identities["word_count"])
```

For each post, it splits every sentence on whitespace to count words and sums them into total. The total per post is stored in `word_count`, and the average is calculated afterwards.

4. Upper Case Character Ratio

```
def get_upper_ratio(self):
    for post in self.data["posts"].values:
        char_count = 0
        upper_count = 0
        for sentence in post:
            for ch in sentence:
                if ch.isalpha():
                    char_count += 1
                    if ch.isupper():
                        upper_count += 1
        if char_count:
            self.basic_identities["upper_ratio"].append(upper_count /
              char_count)
    self.basic_identities["ave_upper_ratio"] = ave(self.
      basic_identities["upper_ratio"])
```

Traverses every character of every sentence, counting alphabetic characters (`char_count`) and, among them, the upper case ones (`upper_count`). The ratio per post is stored; the overall mean is then computed.

5. Readability Metrics

```
def get_readability(self):
    reading_ease = []
    GF_idx = []
    for post in self.data["posts"].values:
        concatenated = post[0]
        for idx in range(1, len(post)):
            concatenated += post[idx]
        reading_ease.append(textstat.flesch_reading_ease(concatenated)
          )
        GF_idx.append(textstat.gunning_fog(concatenated))
    self.basic_identities["reading_ease"] = reading_ease
    self.basic_identities["ave_reading_ease"] = ave(reading_ease)
    self.basic_identities["GF_index"] = GF_idx
    self.basic_identities["ave_GF_index"] = ave(GF_idx)
```

Each post's sentences are concatenated into a single string, after which textstat computes Flesch Reading Ease and Gunning Fog Index. Both per post values and their averages are stored.

The calculation results are printed and displayed.

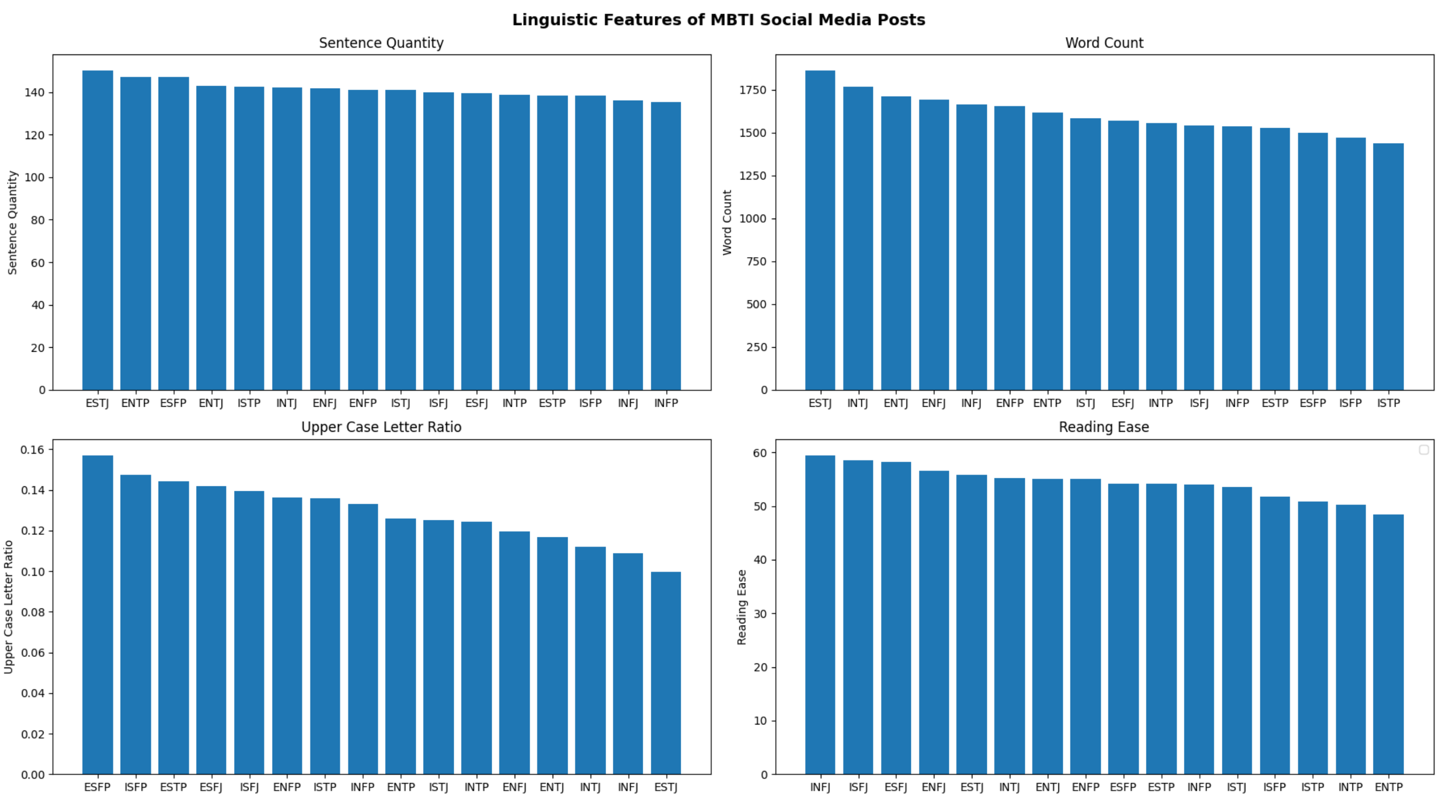
```
mbti_identities={
    T.upper():{
        k:cleaned_data[T].basic_identities[k]
        for k in [
        "ave_sentence_quantity",
        "ave_word_count",
        "ave_upper_ratio",
        "ave_reading_ease",
        "ave_GF_index"
        ]
    }
    for T in MBTI_types
}
mbti_identities=pd.DataFrame(mbti_identities).T
mbti_identities
```

	ave_sentence_quantity	ave_word_count	ave_upper_ratio	ave_reading_ease	ave_GF_index
ISTJ	140.922780	1584.861004	0.125113	53.508301	16.561120
ISFJ	139.890110	1542.766484	0.139261	58.487995	14.762555
INFJ	136.120151	1662.837275	0.108945	59.505421	14.098761
INTJ	142.160051	1768.897567	0.111903	55.228515	15.652458
ISTP	142.599388	1439.327217	0.135959	50.868624	18.008930
ISFP	138.239782	1468.850136	0.147333	51.682698	17.722262
INFP	135.311232	1538.657566	0.133015	54.009626	16.649938
INTP	138.747226	1557.081381	0.124319	50.273724	18.050296
ESTP	138.510000	1528.890000	0.144286	54.100900	16.754100
ESFP	147.137931	1500.454023	0.157108	54.208218	16.571782
ENFP	141.171468	1655.497942	0.136338	55.011289	16.238724
ENTP	147.183709	1617.429809	0.125773	48.465754	18.679307
ESTJ	150.148148	1861.827160	0.099657	55.780123	15.963333
ESFJ	139.666667	1571.104762	0.141718	58.292857	15.020381
ENFJ	141.915058	1690.857143	0.119560	56.535425	15.473456
ENTJ	142.960573	1713.476703	0.116676	55.087849	15.837384

Figure 6.9: Table of Linguistic Style Features of MBTI Social Media Posts

In order to better interpret the underlying patterns in the table, we present descending histograms for visual analysis.

```
column_mapping={
    "ave_sentence_quantity":"Sentence Quantity",
    "ave_word_count":"Word Count",
    "ave_upper_ratio":"Upper Case Letter Ratio",
    "ave_reading_ease":"Reading Ease"
}
fig,axes=plt.subplots(2,2,figsize=(18,10))
axes=axes.flatten()
for i,col in enumerate(mbti_identities.columns[:4]):
    axes[i].bar(mbti_identities[col].sort_values(ascending=False).index, mbti_identities[col].sort_values(ascending=False))
    axes[i].set_ylabel(f"{column_mapping[col]}")
    axes[i].set_title(f"{column_mapping[col]}")
fig.suptitle("Linguistic Features of MBTI Social Media Posts",fontsize=14, fontweight='bold')
plt.legend()
plt.tight_layout()
plt.savefig("final_output/ling_features.png")
plt.show()
```



The histograms allow the observation of surface-level patterns in linguistic styles.

1. Sentence Quantity: The top three MBTI types with the highest average sentence counts are ESTJ (150.1), ENTP (147.2), and ESFP (147.1). In contrast, the bottom three are INFP (135.3), INFJ (136.1), and ISFJ (139.9). This may indicate a correlation between the Extroversion/Introversion type and the extent of verbal expression. However, the overall variation in sentence quantity is relatively minor, suggesting limited explanatory significance.

2. Word Count: ESTJ leads with an average of 1,861.8 words per post, followed by INTJ (1,768.9) and ENTJ (1,713.5). On the other end, ISTP (1,439.3), ISFP (1,468.9) and ESFP (1,500.5) write the fewest words. Overall, Thinking-Judging types (TJs) consistently post at greater length than Perceiving-Feeling types (PFs)

3. Upper-Case Ratio: Posts by ESFP (15.7% uppercase), ISFP (14.7%), and ESTP (14.4%) contain the highest proportion of capital letters, whereas ESTJ (10.0%), INFJ (10.9%), and INTJ (11.2%) post the least. Feeling-Perceiving personalities prefer to use uppercase—perhaps as a tool for emphasis or emotional expression—while Thinking-Judging profiles favor a more uniform casing.

4. Reading Ease: On the Flesch scale, INFJ (59.5), ISFJ (58.5), and ESFJ (58.3) write the most readable posts, whereas ENTP (48.5), INTP (50.3), and ISTP (50.9) produce denser, more complex text. Judging-Feeling types tends for simpler, clearer phrasing, while Perceiving-Thinking types trend toward more intricate language.

Three characteristic can also be observed from the standpoint of MBTI types.

1. ESTJ exhibits the greatest information throughput—leading all types in both mean sentence count and mean word count—whereas ENTP, although nearly as verbose in sentence frequency, records the lowest readability and most negative sentiment, underscoring that sheer output does not translate into a warmer tone.

2. ESFP and ISFP personalities stand out for their high upper-case ratios, corresponding to the commonly held view of ‘FP’ types as spontaneous and emotionally guided.

3. INFJ produces the most readable texts. Meanwhile, they are also among the top three most positive in emotional expression(analyzed in the next section). ENTP produces the least readable and positive texts.

6.2.3 Sentiment Analysis

Sentiment analysis, or opinion mining, is an active area of study in the field of natural language processing that analyzes people's opinions, sentiments, evaluations, attitudes, and emotions via the computational treatment of subjectivity in text. There are various approaches to sentimental analysis. In this assignment, we choose VADER(Valence Aware Dictionary for sEntiment Reasoning) as our tool to analyze the difference of different MBTI personalities in emotional expression due to its superiority in social media context(Kiritchenko, Zhu, & Mohammad, 2014).

We do the following steps to acquire VADER scoring:

```python
analyzer = SentimentIntensityAnalyzer()
overall_vader_score = {'neg': 0.0, 'neu': 0.0, 'pos': 0.0, 'compound': 0.0}
```

- Instantiate NLTK's SentimentIntensityAnalyzer to compute sentiment scores.
- Initialize overall_vader_score as an aggregate dictionary with the four sentiment components (neg, neu, pos, compound), which will be used to assess the text's overall emotional state.

```python
def addup_score_dict(new_dict, base_dict):
    for key in base_dict.keys():
        base_dict[key] += new_dict[key]
```

This function adds the sentiment scores of a sentence (or post) to the target dictionary. It iterates over the keys and sums the corresponding values, allowing either local or global sentiment accumulation.

```python
def process_vader_score(post):
    post_vader_score = {'neg': 0.0, 'neu': 0.0, 'pos': 0.0, 'compound':
      0.0}
    for sentence in post:
        addup_score_dict(analyzer.polarity_scores(sentence), base_dict=
          post_vader_score)
    ave_score_dict(base_dict=post_vader_score, n=len(post))
    addup_score_dict(new_dict=post_vader_score, base_dict=
      overall_vader_score)
    return post_vader_score
```

This nested function analyzes the sentiment of a single post (a list of sentences) through the following steps:

1. Initialize a score container for the post.
2. Traverse each sentence and obtain its sentiment scores with polarity_scores().
3. Accumulate the sentence scores and divide by the number of sentences to compute the post's average sentiment.
4. Add this post's average score to the overall accumulator overall_vader_score.
5. Return the post's sentiment score so it can be stored back in the dataset.

```
self.data["vader_score"] = self.data["posts"].apply(process_vader_score)
```

This line applies process_vader_score to every post in self.data. The returned score dictionaries (neg, neu, pos, compound) are stored in a new column named vader_score.

```
ave_score_dict(overall_vader_score, len(self.data["posts"]))
self.basic_identities["overall_vader_score"] = overall_vader_score
```

After accumulating all post-level scores into overall_vader_score, divide by the total number of posts to obtain the average sentiment. Finally, record this overall score in the basic_identities dictionary under the key "overall_vader_score" as the global emotional characteristic of the personality type.

Having acquired the VADER score, we collect the average sentiment into a chart that is ordered by descending compound score.

```
all_vader_scores={T:cleaned_data[T].basic_identities["overall_vader_score"]
    for T in MBTI_types}
all_vader_scores=pd.DataFrame(all_vader_scores).T
all_vader_scores=all_vader_scores.sort_values(by="compound",ascending=False
  )
all_vader_scores
```

	neg	neu	pos	compound
isfj	0.074714	0.706432	0.179305	0.156459
infj	0.077386	0.706724	0.171925	0.146350
enfj	0.077268	0.706996	0.172352	0.146079
estj	0.075952	0.715335	0.168221	0.135310
esfj	0.079053	0.708754	0.163387	0.126369
enfp	0.081753	0.717179	0.163048	0.125133
intj	0.078297	0.723588	0.157496	0.117230
infp	0.086349	0.708721	0.163746	0.113087
isfp	0.086260	0.709435	0.164908	0.108508
entj	0.082541	0.718532	0.157704	0.106751
istj	0.083894	0.722811	0.153920	0.099382
esfp	0.088280	0.716392	0.156600	0.093912
estp	0.086791	0.719593	0.150472	0.093669
intp	0.086243	0.720133	0.151148	0.090751
istp	0.089930	0.711712	0.150577	0.078989
entp	0.087409	0.723706	0.144536	0.076574

Figure 6.10: Negative, Neutral, Positive and Compound VADER Score arranged in descending compound order

The characteristics of compound VADER score is described.

```
all_vader_scores["compound"].describe()
```

```
count    16.000000
mean      0.113410
std       0.024352
min       0.076574
25%       0.093851
50%       0.110797
75%       0.128604
max       0.156459
Name: compound, dtype: float64
```

Figure 6.11: Description of Compound VADER Scores

From the description, we find that the standard deviation is approximately 0.0243, showing a small overall fluctuation. The range between the most and least positive types spans about

3.3σ, indicating a significant but not extreme variation. Additionally, the standard deviation accounts for only about 20% of the mean, implying a compact and smooth distribution.

The fact that mean is slightly greater than the median suggests a longer tail on the higher end of the scale, which indicates that the distribution exhibits a slight right skew, towards a positive emotional expression. This is supported by a having positive mean value(0.113).

We further visualized the compound results of different MBTI by descending order to find the relationship of MBTI personalities and their sentimental expression.

```
x=all_vader_scores.index
y=all_vader_scores["compound"]
plt.figure(figsize=(10, 6))
plt.bar(x,y)
plt.xlabel("MBTI Types")
plt.ylabel("VADER Compound Score")
plt.title("VADER Compound Score for All MBTI Types",fontsize=14, fontweight
  ='bold')
plt.tight_layout()
plt.savefig("final_output/vader.png")
plt.show()
```

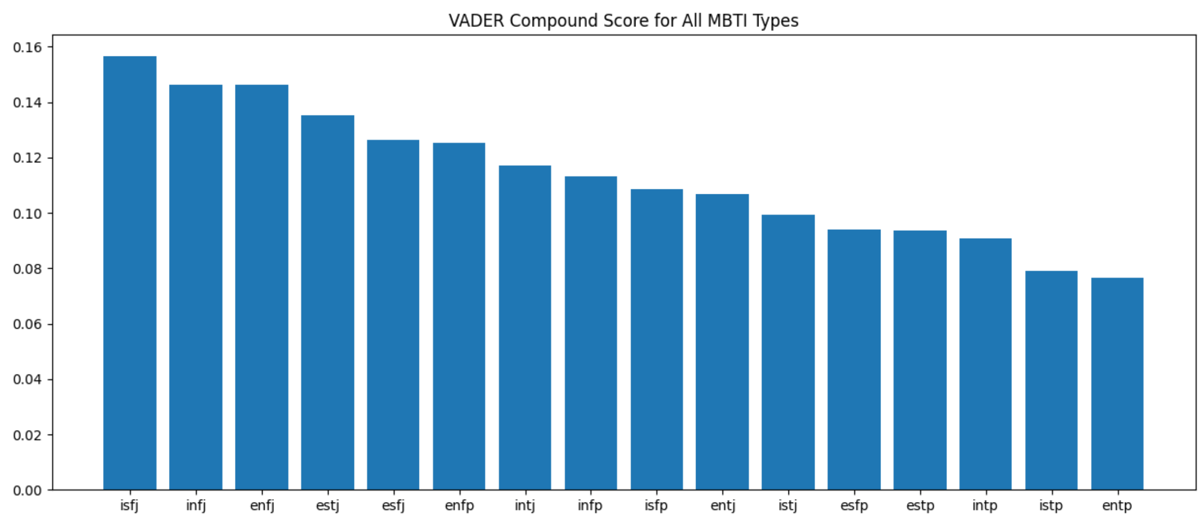


Figure 6.12:

We find that ISFJ, INFJ, ENFJ has the highest compound score, while ISTP, ENTP has the lowest, suggesting personalities with F and J tend to be more positive in emotional expression.

To verify this point, we first decomposed every personality label into its four constituent letters and collected all VADER-compound scores belonging to the same pole of a dichotomy.

If $D \in \{\text{I/E}, \text{S/N}, \text{T/F}, \text{J/P}\}$ and L denotes one pole of D (e.g., $L = \text{F}$ within $D = \text{T/F}$), the group mean is obtained by

$$\bar{x}_L = \frac{1}{n_L} \sum_{k \in L} x_k,$$

where x_k is the compound score of the k-th MBTI type and n_L is the number of types that contain the letter L. Letting

$$\mu = \frac{1}{16} \sum_{i=1}^{16} x_i$$

denote the grand mean across all sixteen personalities, we express the relative contribution of letter L as

$$\Delta_L = \frac{\bar{x}_L - \mu}{\mu} \times 100\%.$$

Applying these definitions to our data we obtain $\bar{x}_\text{F} = 0.126987$, $\bar{x}_\text{T} = 0.099832$, $\bar{x}_\text{J} = 0.129241$, and $\bar{x}_\text{P} = 0.097578$, while the grand mean is $\mu = 0.113410$. Consequently, Feeling exceeds Thinking by $\Delta_{\text{F,T}} \approx 23.9\%$ and Judging exceeds Perceiving by $\Delta_{\text{J,P}} \approx 31.6\%$, whereas the contrasts for Extraversion–Introversion and Intuition–Sensing remain below 3 %. These magnitudes confirm that the third and fourth MBTI letters (T/F, J/P) play a significantly greater role in shaping positive emotional expression, while first two dimension make relatively small contribution.

To find the relation between the four roles—Diplomats (NF), Sentinels (SJ), Analysts (NT) and Explorers (SP)—we test its relative contribution. Writing $G \in \{\text{NF}, \text{SJ}, \text{NT}, \text{SP}\}$ and letting n_G be the number of types in group G ($n_G = 4$), the group score is obtained by

$$\bar{x}_G = \frac{1}{n_G} \sum_{k \in G} x_k, \qquad G \in \{\text{NF}, \text{SJ}, \text{NT}, \text{SP}\},$$

with the grand mean μ defined in equation above. Substituting the individual compound values yields

$$\bar{x}_\text{NF} = 0.133, \qquad \bar{x}_\text{SJ} = 0.129, \qquad \bar{x}_\text{NT} = 0.098, \qquad \bar{x}_\text{SP} = 0.094.$$

To place these figures on the common scale used earlier, we again report the relative deviation

$$\Delta_G = \frac{\bar{x}_G - \mu}{\mu} \times 100\%.$$

Hence

$$\Delta_{\text{NF}} = +17.3\%, \quad \Delta_{\text{SJ}} = +13.7\%, \quad \Delta_{\text{NT}} = -13.6\%, \quad \Delta_{\text{SP}} = -17.1\%.$$

The ranking $\bar{x}_{\text{NF}} > \bar{x}_{\text{SJ}} > \bar{x}_{\text{NT}} > \bar{x}_{\text{SP}}$ mirrors the earlier letter-level findings: Diplomats, combining the affective "F" and the future-oriented "N", produce the warmest tone; Sentinels follow closely, driven by the strongly positive "J" component; Analysts, dominated by the critical "T", adopt a noticeably cooler register; and Explorers, whose "P" preference and playful spontaneity encourage criticism and slang, exhibit the lowest positivity.

```python
def get_sentence_quantity(self):
    for post in self.data["posts"].values:
        self.basic_identities["sentence_quantity"].append(len(post))
    self.basic_identities["ave_sentence_quantity"] = ave(self.
      basic_identities["sentence_quantity"])
```

7 Conclusion

Together, these two strands of evidence allow us to adopt a dialectical stance: MBTI's four dimensions yield reliable behavioral insights, yet its theoretical assumption of strict independence does not fully hold. In other words, MBTI remains a valuable framework—provided its limitations are recognized and its results interpreted with appropriate caution.

On the one hand, according to Pittenger (1993), *"we would expect each factor to be independent of the other factors, inasmuch as the MBTI theory states that each of the four preference dimensions stands alone; questions within one factor should not correlate with questions in the other factors"* (p. 50). However, our empirical tests contradict this assumption: both chi-square analyses and inter-factor correlation matrices reveal that the four MBTI dimensions—E–I, S–N, T–F, and J–P—are not statistically independent, calling into question the theoretical orthogonality upon which MBTI is predicated.

MBTI can play a valuable role in facilitating self-awareness, improving team communication, and guiding career exploration by highlighting broad preference tendencies. It offers a common language for discussing interpersonal differences and can inform coaching, leadership development, and collaborative problem-solving. However, relying too heavily on MBTI carries risks. First, type labels can become “social tags” that oversimplify complex inner worlds and foster stereotypes—for example, assuming every INTJ is cold and rational ignores their potential for empathy and emotional depth. Second, the reliability of any single test result is limited by mood, stress, social-desirability bias, and the test-taker’s context; using one score to choose friends or partners lacks scientific support. Finally, excessive trust in MBTI can open the door to manipulation, as bad actors may exploit type labels to gain trust or social leverage.

In short, MBTI offers valuable directional insight into broad personality tendencies—serving as a springboard for personal growth, team building, and career conversations—but it should never be the sole yardstick for judging an individual. By recognizing its psychometric limitations, avoiding type-based stereotypes, and remaining alert to potential misuse, we can benefit from MBTI’s strengths while steering clear of the pitfalls of oversimplified personality labelling.

References

Fleenor, J. W. (1997). The relationship between the mbti and measures of personality and performance in management groups. *Developing leaders: Research and applications in psychological type and leadership development*, 115–138.

Kiritchenko, S., Zhu, X., & Mohammad, S. (2014, 08). Sentiment analysis of short informal text. *The Journal of Artificial Intelligence Research (JAIR)*, *50*. doi: 10.1613/jair.4272

Yang, Y. (2022). Research on the application of mbti in organization. In *2022 7th international conference on social sciences and economic development (icssed 2022)* (pp. 1751–1754).

Zibran, M. F. (2007). Chi-squared test of independence. *Department of Computer Science, University of Calgary, Alberta, Canada*, *1*(1), 1–7.