Final

April 7, 2025

0.1 Libraries

[3]:
```
import gensim
import pandas as pd
import spacy
import numpy as np
from wefe.word_embedding_model import WordEmbeddingModel
from scipy.stats import pearsonr
from wefe.metrics import RIPA
from wefe.query import Query
import re
from gensim.models import KeyedVectors
import logging
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

[4]:
```
# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s [%(levelname)s] %(message)s'
)
```

0.2 Helper Functions

[5]:
```
def cosine_similarity(v1: np.ndarray, v2: np.ndarray) -> float:
    return np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))

def compute_bias(adj, male_terms, female_terms, model):
    male_mean = np.mean([cosine_similarity(model[adj], model[m]) for m in␣
 ↪male_terms if m in model])
    female_mean = np.mean([cosine_similarity(model[adj], model[f]) for f in␣
 ↪female_terms if f in model])
    return male_mean - female_mean

def compute_individual_bias(
    adjectives,
    male_terms,
```

```
    female_terms,
    model,
    exclude_substrings=True
):
    """
    Computes, for each adjective, the 'bias' difference between the average
    cosine similarity with male terms and the average cosine similarity
    with female terms.

    Parameters
    ----------
    adjectives : list of str
        List of adjectives to be analyzed (already cleaned/lemmatized).
    male_terms : list of str
        Words representing 'masculinity' (e.g., ['man', 'boy', 'father', ...]).
    female_terms : list of str
        Words representing 'femininity' (e.g., ['woman', 'girl', 'lady', ...]).
    model : dict-like of {str -> np.ndarray} or a KeyedVectors-like object
        Your embedding model, where you can check `word in model` and
        retrieve vectors using `model[word]`.
    exclude_substrings : bool, default=True
        Whether to exclude adjectives that contain any of the male or female
        terms as substrings (e.g., "manlike" contains "man").

    Returns
    -------
    pd.DataFrame
        DataFrame with columns:
        ['word', 'male_mean', 'female_mean', 'bias_value'].
        Where 'bias_value' = male_mean - female_mean.
        Rows without sufficient embedding data are skipped.
    """

    # 1) filter out adjectives containing gender terms as substrings
    if exclude_substrings:
        all_target_words = set(male_terms + female_terms)
        def has_target_substring(adj):
            return any(tw in adj for tw in all_target_words)
        adjectives = [adj for adj in adjectives if not␣
 ↪has_target_substring(adj)]

    records = []

    # 2) Loop over each adjective
    for adj in adjectives:
        if adj not in model:
            continue
```

```
        adj_vec = model[adj]

        # Gather cosine similarities with male terms
        male_sims = []
        for m in male_terms:
            if m in model:
                male_sims.append(cosine_similarity(adj_vec, model[m]))

        # Gather cosine similarities with female terms
        female_sims = []
        for f in female_terms:
            if f in model:
                female_sims.append(cosine_similarity(adj_vec, model[f]))

        # Skip if we can't compute both male and female means
        if len(male_sims) == 0 or len(female_sims) == 0:
            continue

        # Compute means
        male_mean = np.mean(male_sims)
        female_mean = np.mean(female_sims)

        # Compute bias
        bias_value = male_mean - female_mean

        records.append({
            "word": adj,
            "male_mean": male_mean,
            "female_mean": female_mean,
            "bias_value": bias_value
        })

    df_bias = pd.DataFrame(records)
    return df_bias

def compute_bias_with_pvalue(adj, male_terms, female_terms, model,␣
 ↪permutations=1000, seed=42):
    """
    Computes the cosine-based gender bias of an adjective relative to male and␣
 ↪female word sets,
    and estimates the statistical significance using a permutation test.

    Parameters
    ----------
    adj : str
        The adjective whose bias is being measured.
```

```
    male_terms : list of str
        A list of words representing the male group (e.g., ["man", "vader",␣
↪"kerel"]).

    female_terms : list of str
        A list of words representing the female group (e.g., ["vrouw",␣
↪"moeder", "meid"]).

    model : KeyedVectors or similar
        A word embedding model that supports word lookup and cosine operations␣
↪(e.g., Word2Vec or FastText).

    permutations : int, default=1000
        The number of random shuffles to perform during the permutation test.

    seed : int, default=42
        Random seed to ensure reproducibility of the permutation results.

    Returns
    -------
    real_bias : float
        The observed bias score, defined as:
        mean_cosine(adj, male_terms) - mean_cosine(adj, female_terms)

    p_value : float
        The estimated probability that a bias of this magnitude (or stronger)␣
↪could occur
        by chance if gender labels were random. Lower values indicate stronger␣
↪significance.

    Description
    -----------
    - Step 1: Compute the real (observed) bias of the adjective using cosine␣
↪similarity.
    - Step 2: Create a combined pool of male and female terms.
    - Step 3: Run `permutations` number of times:
        - Shuffle the gender labels randomly.
        - Split into fake "male" and "female" groups.
        - Compute the permuted bias score.
        - Count how many times the permuted score is greater than or equal to␣
↪the real bias.
    - Step 4: The p-value is the proportion of permutations that produced a␣
↪more extreme bias
              than the observed one.
    """
```

```
    # Ensure reproducibility
    np.random.seed(seed)

    # Step 1: Compute the real cosine-based bias score
    real_bias = compute_bias(adj, male_terms, female_terms, model)

    # Combine all gender terms into one pool to shuffle
    combined_terms = male_terms + female_terms
    num_male = len(male_terms)

    # Counter for how many permuted scores are as extreme as the real one
    extreme_count = 0

    # Step 2: Permutation loop
    for _ in range(permutations):
        np.random.shuffle(combined_terms)

        # Split shuffled terms into permuted male and female groups
        permuted_male = combined_terms[:num_male]
        permuted_female = combined_terms[num_male:]

        # Step 3: Compute bias under this random split
        permuted_bias = compute_bias(adj, permuted_male, permuted_female, model)

        # Count if the permuted bias is more extreme than the observed one
        if abs(permuted_bias) >= abs(real_bias):
            extreme_count += 1

    # Step 4: Compute the p-value as the proportion of extreme permutations
    p_value = extreme_count / permutations

    return real_bias, p_value

def tag_bias_agreement(row, alpha=0.05):
    sig_w2v = row['p_value_w2v'] < alpha
    sig_ft  = row['p_value_ft'] < alpha
    same_sign = np.sign(row['bias_w2v']) == np.sign(row['bias_ft'])

    if sig_w2v and sig_ft:
        if same_sign:
            return "Significant in both (agree)"
        else:
            return "Significant in both (oppose)"
    elif sig_w2v:
        return "Only Word2Vec"
    elif sig_ft:
```

```
        return "Only FastText"
    else:
        return "Non-significant"
```

0.3 Embedding Models

[6]:
```
################################################################################
# 1) Load FastText embeddings
################################################################################
logging.info("Loading Fasttext embeddings with Gensim from file...")
nl_embeddings = gensim.models.KeyedVectors.load_word2vec_format(
    "/Users/matthijstentije/University/MSc_Data-Science/Thesis/
 ↪MSc_Data_Science_Thesis/data/cc.nl.300.vec.gz",
    binary=False
)
logging.info("Fasttext embeddings loaded.")

# Convert to WEFE-compatible format
fasttext_model = WordEmbeddingModel(nl_embeddings, "Dutch FastText")
logging.info("Fasttext Embeddings Model Created.")

################################################################################
# 2) Load Word2Vec model
################################################################################
logging.info("Loading Word2Vec model from file...")
model_path = "/Users/matthijstentije/University/MSc_Data-Science/Thesis/
 ↪MSc_Data_Science_Thesis/data/sonar-320.txt"
model_w2v = KeyedVectors.load_word2vec_format(model_path, binary=False)
logging.info("Word2Vec Model loaded successfully.")
```

```
2025-04-07 12:55:22,145 [INFO] Loading Fasttext embeddings with Gensim from
file…
2025-04-07 12:55:22,146 [INFO] loading projection weights from
/Users/matthijstentije/University/MSc_Data-
Science/Thesis/MSc_Data_Science_Thesis/data/cc.nl.300.vec.gz
2025-04-07 12:55:22,146 [INFO] loading projection weights from
/Users/matthijstentije/University/MSc_Data-
Science/Thesis/MSc_Data_Science_Thesis/data/cc.nl.300.vec.gz
2025-04-07 12:58:03,174 [INFO] KeyedVectors lifecycle event {'msg': 'loaded
(2000000, 300) matrix of type float32 from
/Users/matthijstentije/University/MSc_Data-
Science/Thesis/MSc_Data_Science_Thesis/data/cc.nl.300.vec.gz', 'binary': False,
'encoding': 'utf8', 'datetime': '2025-04-07T12:58:03.174444', 'gensim': '4.3.3',
'python': '3.12.1 (v3.12.1:2305ca5144, Dec  7 2023, 17:23:38) [Clang 13.0.0
(clang-1300.0.29.30)]', 'platform': 'macOS-15.3.2-arm64-arm-64bit', 'event':
'load_word2vec_format'}
2025-04-07 12:58:03,174 [INFO] Fasttext embeddings loaded.
```

```
2025-04-07 12:58:03,175 [INFO] Fasttext Embeddings Model Created.
2025-04-07 12:58:03,175 [INFO] Loading Word2Vec model from file…
2025-04-07 12:58:03,175 [INFO] loading projection weights from
/Users/matthijstentije/University/MSc_Data-
Science/Thesis/MSc_Data_Science_Thesis/data/sonar-320.txt
2025-04-07 12:58:58,172 [INFO] KeyedVectors lifecycle event {'msg': 'loaded
(626711, 320) matrix of type float32 from
/Users/matthijstentije/University/MSc_Data-
Science/Thesis/MSc_Data_Science_Thesis/data/sonar-320.txt', 'binary': False,
'encoding': 'utf8', 'datetime': '2025-04-07T12:58:58.172136', 'gensim': '4.3.3',
'python': '3.12.1 (v3.12.1:2305ca5144, Dec  7 2023, 17:23:38) [Clang 13.0.0
(clang-1300.0.29.30)]', 'platform': 'macOS-15.3.2-arm64-arm-64bit', 'event':
'load_word2vec_format'}
2025-04-07 12:58:58,172 [INFO] Word2Vec Model loaded successfully.
```

0.4 Extracting Adjectives + Spacy

[7]:
```
###############################################################################
# 3) Load spaCy for Dutch, define function to extract adjectives
###############################################################################
nlp = spacy.load('nl_core_news_lg')

def extract_adjectives_from_csv(file_path):
    """
    Leest een CSV-bestand in, parse elke phrase met spaCy,
    en retourneert unieke gelemmatiseerde bijvoeglijke naamwoorden.
    """
    logging.info(f"Loading CSV file: {file_path}")
    try:
        df = pd.read_csv(file_path, delimiter=';', usecols=[0],␣
↪names=["Group"], header=0)
        logging.info(f"CSV loaded successfully with shape: {df.shape}")
    except Exception as e:
        logging.error(f"Failed to load CSV file: {e}")
        raise

    df.dropna(subset=["Group"], inplace=True)
    logging.info(f"Dropped NaN rows. Remaining phrases: {len(df)}")

    adjectives = []
    logging.info("Starting POS tagging and lemmatization...")
    for idx, phrase in enumerate(df["Group"]):
        doc = nlp(phrase)
        for token in doc:
            if token.pos_ == "ADJ" and token.is_alpha:
                adjectives.append(token.lemma_.lower())
        if idx % 1000 == 0 and idx > 0:
```

```
            logging.info(f"Processed {idx} phrases...")

    unique_adjectives = list(dict.fromkeys(adjectives))
    logging.info(f"Extracted {len(unique_adjectives)} unique adjectives.")
    return unique_adjectives

# ---- Use the function ----
csv_file_path = "/Users/matthijstentije/University/MSc_Data-Science/Thesis/
 ↪MSc_Data_Science_Thesis/data/Corpus_Hedendaags_Nederlands_Adjectives.csv"
adjectives = extract_adjectives_from_csv(csv_file_path)
```

```
2025-04-07 12:59:00,788 [INFO] Loading CSV file:
/Users/matthijstentije/University/MSc_Data-Science/Thesis/MSc_Data_Science_Thesi
s/data/Corpus_Hedendaags_Nederlands_Adjectives.csv
2025-04-07 12:59:00,809 [INFO] CSV loaded successfully with shape: (19242, 1)
2025-04-07 12:59:00,815 [INFO] Dropped NaN rows. Remaining phrases: 19239
2025-04-07 12:59:00,815 [INFO] Starting POS tagging and lemmatization…
2025-04-07 12:59:02,449 [INFO] Processed 1000 phrases…
2025-04-07 12:59:04,042 [INFO] Processed 2000 phrases…
2025-04-07 12:59:05,668 [INFO] Processed 3000 phrases…
2025-04-07 12:59:07,271 [INFO] Processed 4000 phrases…
2025-04-07 12:59:08,864 [INFO] Processed 5000 phrases…
2025-04-07 12:59:10,467 [INFO] Processed 6000 phrases…
2025-04-07 12:59:12,072 [INFO] Processed 7000 phrases…
2025-04-07 12:59:13,660 [INFO] Processed 8000 phrases…
2025-04-07 12:59:15,241 [INFO] Processed 9000 phrases…
2025-04-07 12:59:16,837 [INFO] Processed 10000 phrases…
2025-04-07 12:59:18,419 [INFO] Processed 11000 phrases…
2025-04-07 12:59:19,999 [INFO] Processed 12000 phrases…
2025-04-07 12:59:21,598 [INFO] Processed 13000 phrases…
2025-04-07 12:59:23,185 [INFO] Processed 14000 phrases…
2025-04-07 12:59:24,767 [INFO] Processed 15000 phrases…
2025-04-07 12:59:26,359 [INFO] Processed 16000 phrases…
2025-04-07 12:59:27,947 [INFO] Processed 17000 phrases…
2025-04-07 12:59:29,525 [INFO] Processed 18000 phrases…
2025-04-07 12:59:31,104 [INFO] Processed 19000 phrases…
2025-04-07 12:59:31,490 [INFO] Extracted 2938 unique adjectives.
```

[8]:

```
adjectives_w2v = {w for w in adjectives if w in model_w2v}
adjectives_ft  = {w for w in adjectives if w in fasttext_model}


#  Take the intersection of the two sets
union_vocab = adjectives_w2v.intersection(adjectives_ft)
print(f"Number of adjectives in Word2Vec: {len(adjectives_w2v)}")
print(f"Number of adjectives in FastText : {len(adjectives_ft)}")
print(f"Total in intersection (Word2Vec   FastText): {len(union_vocab)}")
```

```
# Exclude 'target_words' from the intersection
target_words = [
    "man", "kerel", "jongen", "vader", "zoon", "vent", "gast", "meneer",
    "opa", "oom", "vrouw", "dame", "meisje", "moeder", "dochter",
    "tante", "oma", "mevrouw", "meid",
]

filtered_adjectives = [
    adj for adj in union_vocab
    if not any(tw in adj for tw in target_words)
]
print(f"Remaining adjectives in the intersection after filtering target words:␣
 ↪{len(filtered_adjectives)}")
```

```
Number of adjectives in Word2Vec: 2777
Number of adjectives in FastText : 2787
Total in intersection (Word2Vec   FastText): 2765
Remaining adjectives in the intersection after filtering target words: 2741
```

0.5 Cosine Similarity Bias

[9]:
```
# Define your reference sets (as in your message)
MALE_WORDS = ["man", "kerel", "jongen", "vader", "zoon", "vent", "gast",␣
 ↪"meneer", "opa", "oom"]
FEMALE_WORDS = ["vrouw", "dame", "meisje", "moeder", "dochter", "tante", "oma",␣
 ↪"mevrouw", "meid"]
```

[10]:
```
# STEP 1 - WORD2VEC
logging.info("Step 1 (W2V): Computing raw cosine biases for all adjectives...")

df_indiv_bias_w2v = compute_individual_bias(
    adjectives=filtered_adjectives,
    male_terms=MALE_WORDS,
    female_terms=FEMALE_WORDS,
    model=model_w2v,
    exclude_substrings=True  # Optional, avoids words like "mannetje"
)

logging.info(f"Word2Vec: Computed raw bias for {len(df_indiv_bias_w2v)}␣
 ↪adjectives.")


# STEP 1 - FASTTEXT
logging.info("Step 1 (FastText): Computing raw cosine biases for all adjectives.
 ↪..")
```

```
df_indiv_bias_ft = compute_individual_bias(
    adjectives=filtered_adjectives,
    male_terms=MALE_WORDS,
    female_terms=FEMALE_WORDS,
    model=fasttext_model,
    exclude_substrings=True
)

logging.info(f"FastText: Computed raw bias for {len(df_indiv_bias_ft)}␣
↪adjectives.")
```

```
2025-04-07 12:59:31,526 [INFO] Step 1 (W2V): Computing raw cosine biases for all
adjectives…
2025-04-07 12:59:31,733 [INFO] Word2Vec: Computed raw bias for 2741 adjectives.
2025-04-07 12:59:31,733 [INFO] Step 1 (FastText): Computing raw cosine biases
for all adjectives…
2025-04-07 12:59:31,935 [INFO] FastText: Computed raw bias for 2741 adjectives.
```

[11]:
```
# STEP 2 - WORD2VEC
df_indiv_bias_w2v['abs_bias'] = df_indiv_bias_w2v['bias_value'].abs()
top_bias_words_w2v = df_indiv_bias_w2v.sort_values('abs_bias',␣
↪ascending=False)['word'].tolist()

# STEP 2 - FASTTEXT
df_indiv_bias_ft['abs_bias'] = df_indiv_bias_ft['bias_value'].abs()
top_bias_words_ft = df_indiv_bias_ft.sort_values('abs_bias',␣
↪ascending=False)['word'].tolist()

logging.info(f"most biased words (W2V):")
print(top_bias_words_w2v[:10])
logging.info(f"Most biased words (FastText):")
print(top_bias_words_ft[:10])
```

```
2025-04-07 12:59:31,944 [INFO] most biased words (W2V):
2025-04-07 12:59:31,945 [INFO] Most biased words (FastText):
```

```
['lesbisch', 'blond', 'achtjarig', 'beeldschoon', 'zwanger', 'ongepland',
'bloedmooie', 'beeldig', 'ongehuwd', 'kinderloos']
['snoezig', 'beeldschoon', 'zwanger', 'mollig', 'tuttig', 'sensueel',
'genitaal', 'feminien', 'lieftallig', 'superschattig']
```

[18]:
```
df_indiv_bias_w2v
```

[18]:

	word	male_mean	female_mean	bias_value	abs_bias
0	failliet	0.211464	0.197633	0.013831	0.013831
1	waakzaam	0.170601	0.151696	0.018905	0.018905
2	kwiek	0.368230	0.401834	-0.033604	0.033604

```
3       dyslectisch   0.353797     0.348445    0.005352  0.005352
4      dubbelzinnig   0.234209     0.243613   -0.009403  0.009403
…               …         …            …          …          …
2736       mexicaan   0.318820     0.285486    0.033333  0.033333
2737   evolutionair   0.250724     0.223318    0.027406  0.027406
2738      periodiek   0.116918     0.126628   -0.009710  0.009710
2739     overbelast   0.156835     0.163666   -0.006831  0.006831
2740  bedrijfsmatig   0.114794     0.104493    0.010301  0.010301

[2741 rows x 5 columns]
```

[ ]:
```python
# STEP 3 - WORD2VEC
logging.info(f"Step 3 (W2V): Running permutation p-value test on adjectives...")
results_w2v = []

for word in filtered_adjectives:
    try:
        bias, pval = compute_bias_with_pvalue(word, MALE_WORDS, FEMALE_WORDS,␣
 ↪model_w2v)
        results_w2v.append({'word': word, 'bias': bias, 'p_value': pval})
    except Exception as e:
        logging.warning(f"Word2Vec error on '{word}': {e}")

df_bias_sig_w2v = pd.DataFrame(results_w2v).sort_values('p_value')
logging.info(f"Word2Vec: Finished permutation testing for {len(results_w2v)}␣
 ↪words.")
print("\n=== Top Word2Vec Results (sorted by p-value) ===")
print(df_bias_sig_w2v.head(10))


# STEP 3 - FASTTEXT
logging.info(f"Step 3 (FastText): Running permutation p-value test on␣
 ↪adjectives...")
results_ft = []

for word in filtered_adjectives:
    try:
        bias, pval = compute_bias_with_pvalue(word, MALE_WORDS, FEMALE_WORDS,␣
 ↪fasttext_model)
        results_ft.append({'word': word, 'bias': bias, 'p_value': pval})
    except Exception as e:
        logging.warning(f"  FastText error on '{word}': {e}")

df_bias_sig_ft = pd.DataFrame(results_ft).sort_values('p_value')
logging.info(f"FastText: Finished permutation testing for {len(results_ft)}␣
 ↪words.")
print("\n=== Top FastText Results (sorted by p-value) ===")
```

```
print(df_bias_sig_ft.head(10))
```

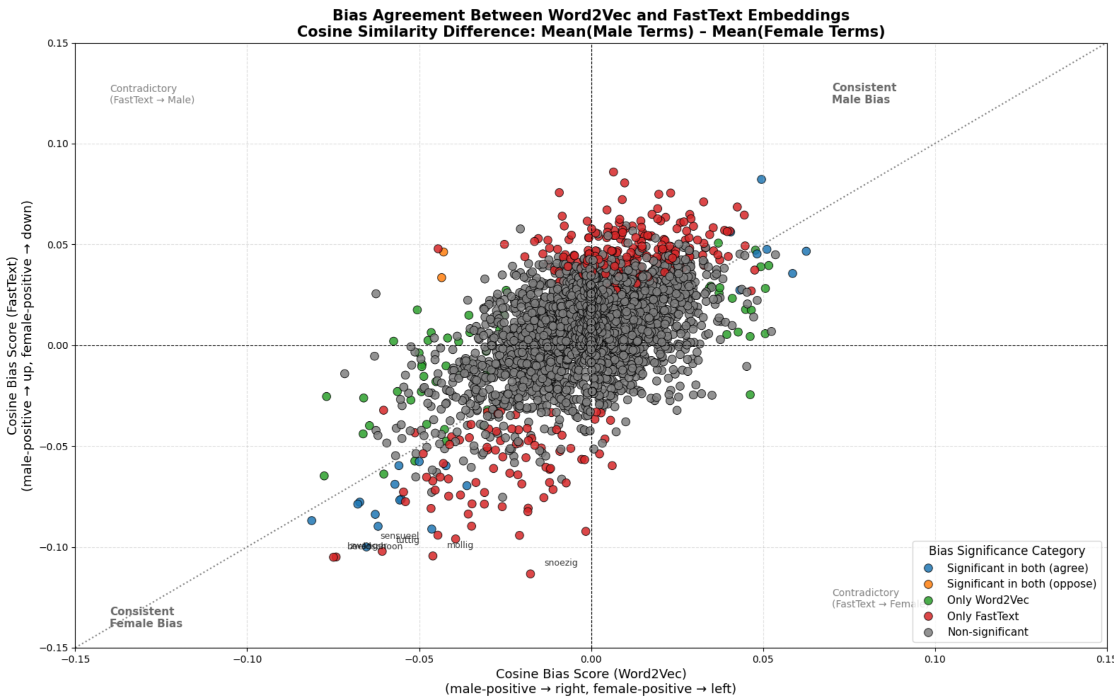
```
2025-04-07 13:39:01,700 [INFO] Step 3 (W2V): Running permutation p-value test on
adjectives…
```

0.5.1 Metrics

```
[ ]: # Merge on word
df_compare = pd.merge(
    df_bias_sig_w2v.rename(columns={'bias': 'bias_w2v', 'p_value':␣
 ↪'p_value_w2v'}),
    df_bias_sig_ft.rename(columns={'bias': 'bias_ft', 'p_value': 'p_value_ft'}),
    on='word',
    suffixes=('', '_ft')
)

# Tag each row
df_compare['tag'] = df_compare.apply(tag_bias_agreement, axis=1)

# Set figure
plt.figure(figsize=(16, 10))

# Color palette
palette = {
    "Significant in both (agree)": "#1f77b4",    # blue
    "Significant in both (oppose)": "#ff7f0e",   # orange
    "Only Word2Vec": "#2ca02c",                  # green
    "Only FastText": "#d62728",                  # red
    "Non-significant": "#7f7f7f"                 # gray
}

# Scatterplot
sns.scatterplot(
    data=df_compare,
    x='bias_w2v',
    y='bias_ft',
    hue='tag',
    palette=palette,
    s=70,
    edgecolor='black',
    alpha=0.85
)

# Axis labels
plt.xlabel("Cosine Bias Score (Word2Vec)\n(male-positive → right,␣
 ↪female-positive → left)", fontsize=13)
```

```
plt.ylabel("Cosine Bias Score (FastText)\n(male-positive → up, female-positive → down)", fontsize=13)

# Title
plt.title(
    "Bias Agreement Between Word2Vec and FastText Embeddings\n"
    "Cosine Similarity Difference: Mean(Male Terms) - Mean(Female Terms)",
    fontsize=15,
    weight='bold'
)

# Reference lines
plt.axhline(0, color='black', linestyle='--', linewidth=0.8)
plt.axvline(0, color='black', linestyle='--', linewidth=0.8)
plt.plot([-0.15, 0.15], [-0.15, 0.15], linestyle=':', color='gray')  # y = x

# Quadrant annotations (adjusted to fit new x/y limits)
plt.text(0.07, 0.12, "Consistent\nMale Bias", fontsize=11, weight='bold', color='dimgray')
plt.text(-0.14, 0.12, "Contradictory\n(FastText → Male)", fontsize=10, color='gray')
plt.text(-0.14, -0.14, "Consistent\nFemale Bias", fontsize=11, weight='bold', color='dimgray')
plt.text(0.07, -0.13, "Contradictory\n(FastText → Female)", fontsize=10, color='gray')

# Label extreme words (bias > 0.1 in either model)
strong = df_compare[
    (df_compare['bias_w2v'].abs() > 0.1) |
    (df_compare['bias_ft'].abs() > 0.1)
]

for _, row in strong.iterrows():
    plt.text(
        row['bias_w2v'] + 0.002,
        row['bias_ft'] + 0.002,
        row['word'],
        fontsize=9,
        color='black',
        alpha=0.8
    )

# Legend (clean and ordered)
handles, labels = plt.gca().get_legend_handles_labels()
order = [
    "Significant in both (agree)",
    "Significant in both (oppose)",
```

```
    "Only Word2Vec",
    "Only FastText",
    "Non-significant"
]
ordered = sorted(zip(labels, handles), key=lambda x: order.index(x[0]))
labels, handles = zip(*ordered)

plt.legend(
    handles, labels,
    title="Bias Significance Category",
    loc='lower right',
    fontsize=11,
    title_fontsize=12,
    frameon=True
)

# Tidy grid & layout
plt.grid(True, linestyle='--', alpha=0.4)
plt.xlim(-0.15, 0.15)
plt.ylim(-0.15, 0.15)
plt.tight_layout()
plt.show()
```



[ ]:
```python
corr, pval = pearsonr(df_compare['bias_w2v'], df_compare['bias_ft'])
print(f"Correlation (Word2Vec vs. FastText) = {corr:.3f} (p = {pval:.4g})")
```

```
Correlation (Word2Vec vs. FastText) = 0.561 (p = 1.632e-227)
```

0.5.2 Z-scores

[ ]:
```python
# STEP 1 - Z-scores for Word2Vec
mean_w2v = df_bias_sig_w2v['bias'].mean()
std_w2v = df_bias_sig_w2v['bias'].std()

df_bias_sig_w2v['z_score_w2v'] = (df_bias_sig_w2v['bias'] - mean_w2v) / std_w2v

# STEP 1 - Z-scores for FastText
mean_ft = df_bias_sig_ft['bias'].mean()
std_ft = df_bias_sig_ft['bias'].std()

df_bias_sig_ft['z_score_ft'] = (df_bias_sig_ft['bias'] - mean_ft) / std_ft
```

[ ]:
```python
# Merge on shared adjectives
df_z_compare = pd.merge(
    df_bias_sig_w2v[['word', 'z_score_w2v']],
    df_bias_sig_ft[['word', 'z_score_ft']],
    on='word'
)

# Select top N based on absolute average Z-score
df_z_compare['avg_abs_z'] = (df_z_compare['z_score_w2v'].abs() +
 ↪df_z_compare['z_score_ft'].abs()) / 2
df_top = df_z_compare.sort_values('avg_abs_z', ascending=False).head(15)

print("\n=== Top 15 Biased Words Across Both Models (by average Z) ===")
print(df_top[['word', 'z_score_w2v', 'z_score_ft']])
```

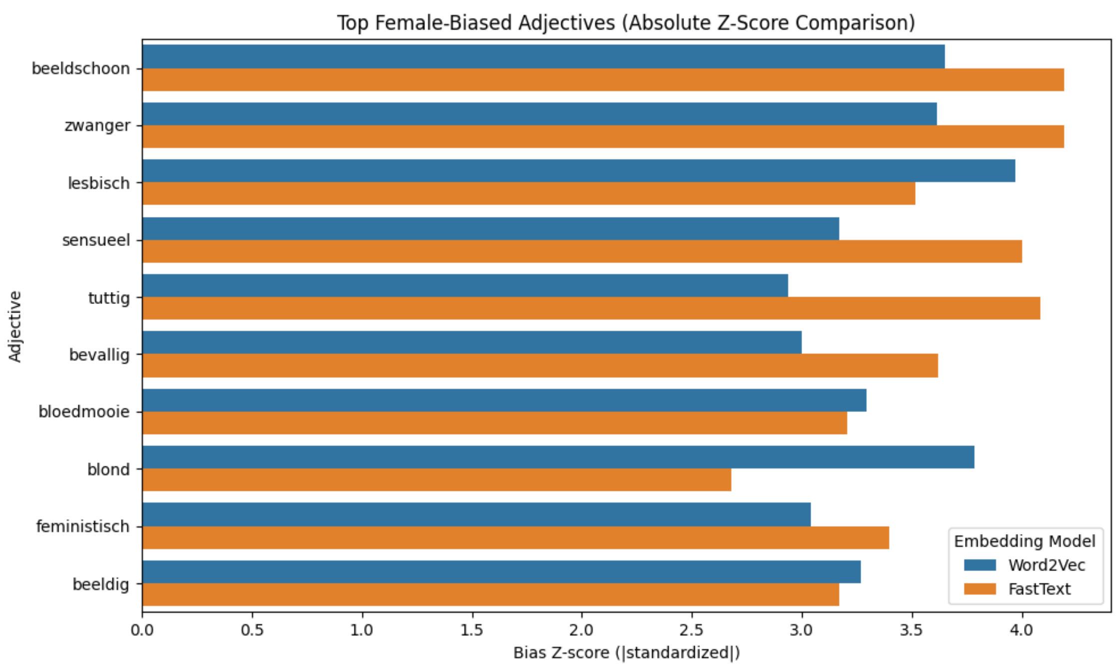
```
=== Top 15 Biased Words Across Both Models (by average Z) ===
             word  z_score_w2v  z_score_ft
120   beeldschoon    -3.649789   -4.194528
83        zwanger    -3.612009   -4.190137
4        lesbisch    -3.968224   -3.513527
47       sensueel    -3.168525   -4.002907
92         tuttig    -2.938378   -4.084665
25       bevallig    -2.998670   -3.619346
26      bloedmooie   -3.295097   -3.206470
22          blond    -3.785365   -2.681660
18    feministisch   -3.039899   -3.397320
13        beeldig    -3.266445   -3.169605
157        mollig    -2.198890   -4.170707
```

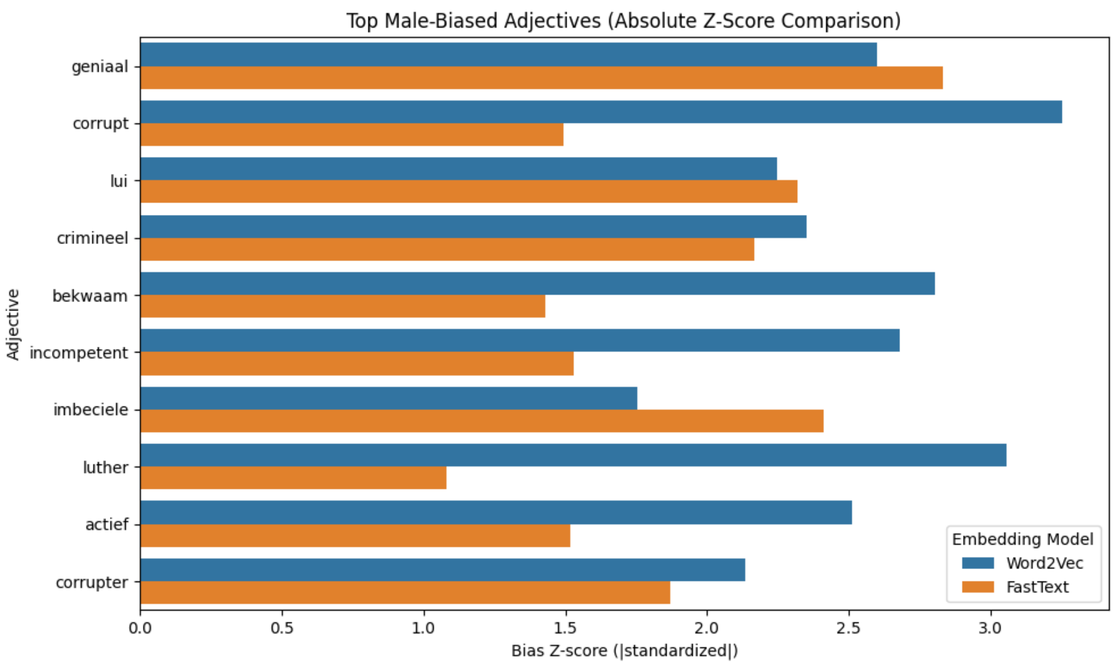
```
346    lieftallig   -2.125926   -3.783992
8           zedig   -2.215161   -3.671073
77     goudblonde   -2.681639   -3.131600
40     glamoureus   -2.660143   -3.115621
```

[ ]:
```python
# Filter separately
df_male = df_z_compare[
    (df_z_compare['z_score_w2v'] > 0) & (df_z_compare['z_score_ft'] > 0)
].copy()

df_female = df_z_compare[
    (df_z_compare['z_score_w2v'] < 0) & (df_z_compare['z_score_ft'] < 0)
].copy()

# top-N limit
TOP_N = 10
df_male = df_male.sort_values('avg_abs_z', ascending=False).head(TOP_N)
df_female = df_female.sort_values('avg_abs_z', ascending=False).head(TOP_N)
```

[ ]:
```python
def plot_bias_barplot_abs(df_subset, title):
    """
    Plot absolute Z-scores for bias (e.g., for female-biased words),
    so all bars go left to right.
    """
    df_plot = df_subset.copy()

    # Take absolute Z-scores
    df_plot['z_score_w2v'] = df_plot['z_score_w2v'].abs()
    df_plot['z_score_ft'] = df_plot['z_score_ft'].abs()

    # Sort by average
    df_plot['avg_z'] = (df_plot['z_score_w2v'] + df_plot['z_score_ft']) / 2
    df_plot = df_plot.sort_values('avg_z', ascending=False)

    df_melted = df_plot.melt(
        id_vars='word',
        value_vars=['z_score_w2v', 'z_score_ft'],
        var_name='Model',
        value_name='Z-score'
    )
    df_melted['Model'] = df_melted['Model'].map({
        'z_score_w2v': 'Word2Vec',
        'z_score_ft': 'FastText'
    })
    df_melted['word'] = pd.Categorical(df_melted['word'],
 ↪categories=df_plot['word'], ordered=True)
```

```
    plt.figure(figsize=(10, 6))
    sns.barplot(
        data=df_melted,
        y='word',
        x='Z-score',
        hue='Model',
        orient='h'
    )
    plt.title(title)
    plt.xlabel("Bias Z-score (|standardized|)")
    plt.ylabel("Adjective")
    plt.legend(title="Embedding Model", loc='lower right')
    plt.tight_layout()
    plt.show()

# Call the updated function for female-biased words
plot_bias_barplot_abs(df_female, "Top Female-Biased Adjectives (Absolute␣
↪Z-Score Comparison)")
plot_bias_barplot_abs(df_male, "Top Male-Biased Adjectives (Absolute Z-Score␣
↪Comparison)")
```





0.6 RIPA

[ ]:
```
class GensimDutchEmbeddingModel(WordEmbeddingModel):
    def __init__(self, keyed_vectors):
        super().__init__(wv=keyed_vectors)
w2v_model = GensimDutchEmbeddingModel(model_w2v)
```

[ ]:
```
# Define the query
query = Query(
    target_sets=[
        ["man", "kerel", "jongen", "vader", "zoon", "vent", "meneer", "opa",␣
 ↪"oom"],
        ["vrouw", "dame", "meisje", "moeder", "dochter", "tante", "oma",␣
 ↪"mevrouw", "meid"]],
    attribute_sets=[adjectives],
    target_sets_names=["Male Terms", "Female Terms"],
    attribute_sets_names=["Adjectives"],
)

ripa = RIPA()
result_ripa_w2v = ripa.run_query(query, w2v_model)
result_ripa_ft = ripa.run_query(query, fasttext_model)
```

[ ]:
```
# 'result["word_values"]' {woord: {'mean': x, 'std': y}, ...}
df_ripa_w2v = pd.DataFrame({
```

```
    'Word': result_ripa_w2v["word_values"].keys(),
    'Mean Score': [val['mean'] for val in result_ripa_w2v["word_values"].
 ↪values()],
    'Std Dev': [val['std'] for val in result_ripa_w2v["word_values"].values()],
})

search_words = ["sterk", "zacht", "moedig", "emotioneel", "dominant",
                "zorgzaam", "aardig", "knap", "schattig"]

# Sorteer op Mean Score (die RIPA per woord toekent) en bekijk
df_ripa_w2v = df_ripa_w2v.sort_values(by="Mean Score", ascending=False).
 ↪reset_index(drop=True)

df_ripa_ft = pd.DataFrame({
    'Word': result_ripa_ft["word_values"].keys(),
    'Mean Score': [val['mean'] for val in result_ripa_ft["word_values"].
 ↪values()],
    'Std Dev': [val['std'] for val in result_ripa_ft["word_values"].values()],
})


# Sorteer op Mean Score (die RIPA per woord toekent) en bekijk
df_ripa_w2v = df_ripa_w2v.sort_values(by="Mean Score", ascending=False).
 ↪reset_index(drop=True)
df_ripa_ft = df_ripa_ft.sort_values(by="Mean Score", ascending=False).
 ↪reset_index(drop=True)
```

[ ]:
```
# Z-score
mean_of_scores_w2v = df_ripa_w2v["Mean Score"].mean()
std_of_scores_w2v = df_ripa_w2v["Mean Score"].std()
df_ripa_w2v["Z-Score"] = (df_ripa_w2v["Mean Score"] - mean_of_scores_w2v) /␣
 ↪std_of_scores_w2v
df_ripa_w2v = df_ripa_w2v.sort_values("Z-Score", ascending=False).
 ↪reset_index(drop=True)


mean_of_scores_ft = df_ripa_ft["Mean Score"].mean()
std_of_scores_ft = df_ripa_ft["Mean Score"].std()
df_ripa_ft["Z-Score"] = (df_ripa_ft["Mean Score"] - mean_of_scores_ft) /␣
 ↪std_of_scores_ft
df_ripa_ft = df_ripa_ft.sort_values("Z-Score", ascending=False).
 ↪reset_index(drop=True)
```

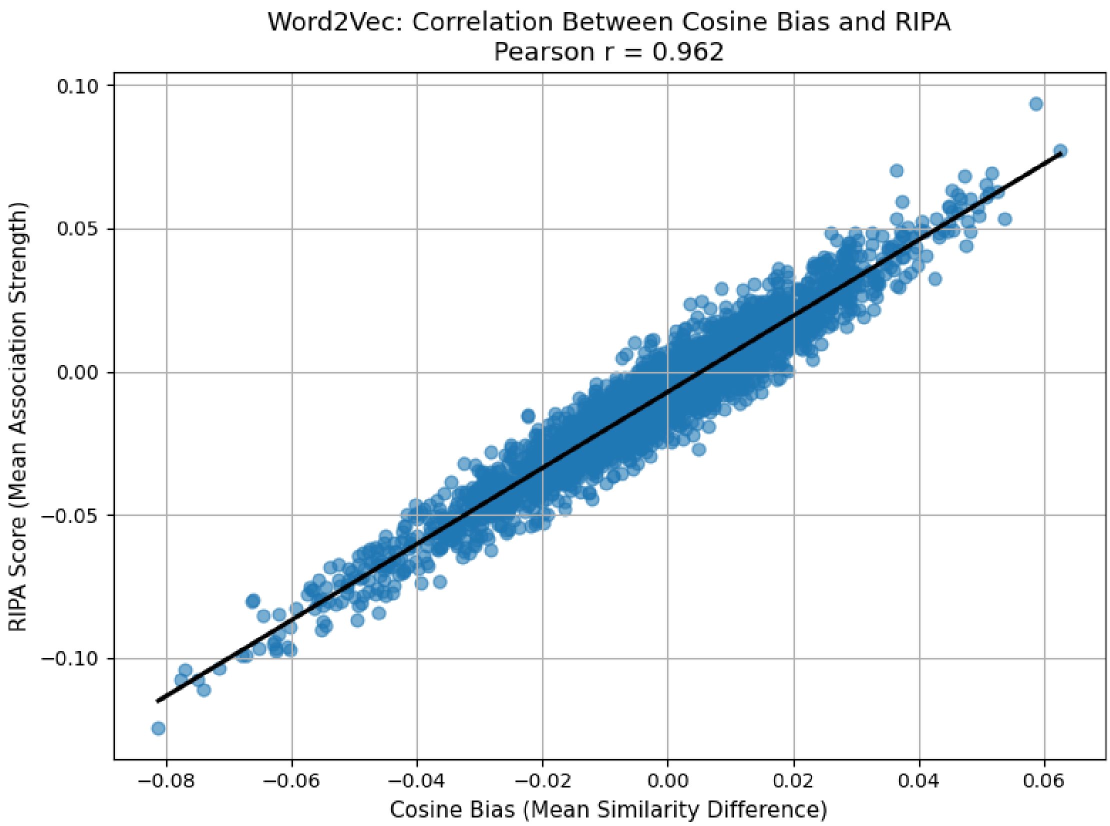
[ ]:
```
def prepare_bias_comparison(df_cosine, df_ripa):
    """
```

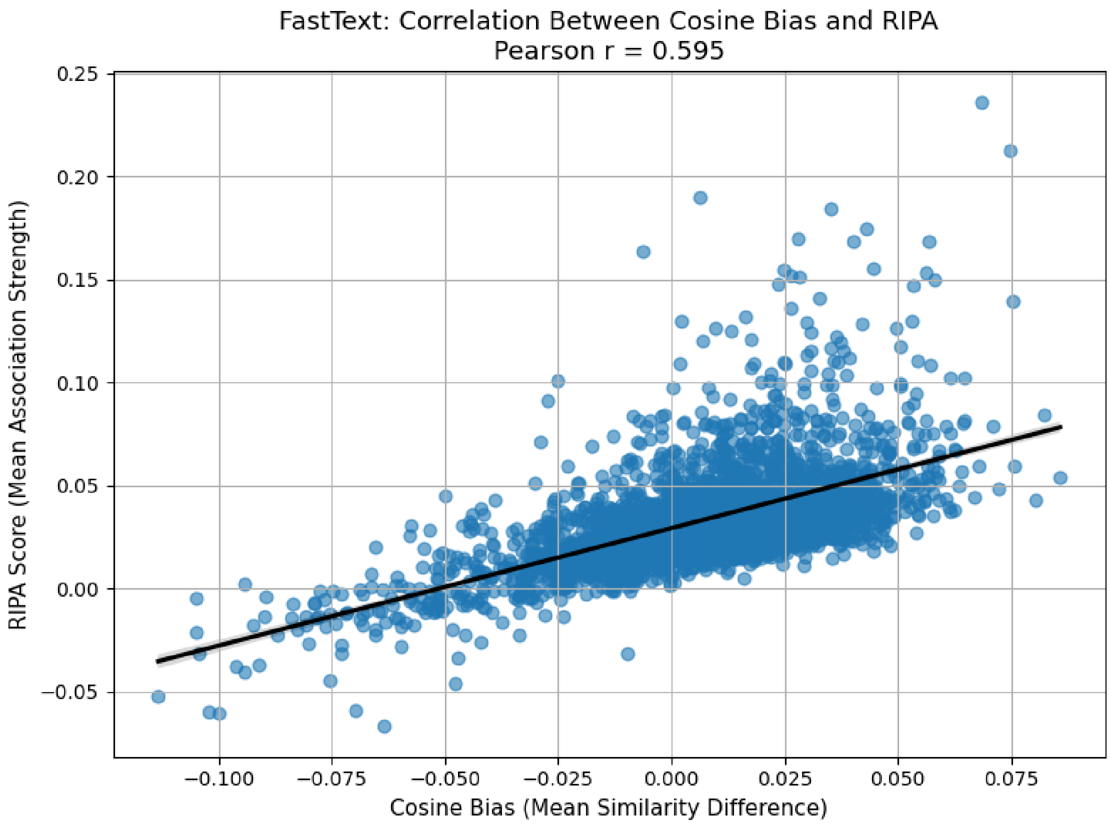
```
    Merge cosine similarity bias with RIPA scores, compute Z-scores, and return␣
↪full merged DataFrame.
    """
    df = pd.merge(
        df_cosine,
        df_ripa[['Word', 'Mean Score']],
        left_on='word',
        right_on='Word',
        how='inner'
    ).rename(columns={'Mean Score': 'RIPA_score'})

    # Z-score normalize both metrics
    df['cosine_bias_z'] = (df['bias'] - df['bias'].mean()) / df['bias'].std()
    df['ripa_z'] = (df['RIPA_score'] - df['RIPA_score'].mean()) /␣
↪df['RIPA_score'].std()

    return df.drop(columns='Word')
# Assume you have:
# - df_bias_pval_w2v
# - df_ripa_w2v
# - df_bias_pval_ft
# - df_ripa_ft

df_combined_w2v = prepare_bias_comparison(df_bias_sig_w2v, df_ripa_w2v)
df_combined_ft  = prepare_bias_comparison(df_bias_sig_ft, df_ripa_ft)
```

[ ]:

```
import scipy.stats as stats
def plot_bias_correlation(df, model_name):
    corr, p_val = stats.pearsonr(df['bias'], df['RIPA_score'])

    plt.figure(figsize=(8, 6))
    sns.regplot(
        x='bias',
        y='RIPA_score',
        data=df,
        scatter_kws={'alpha': 0.6, 's': 40},
        line_kws={'color': 'black'}
    )
    plt.title(
        f"{model_name}: Correlation Between Cosine Bias and RIPA\n"
        f"Pearson r = {corr:.3f}",
        fontsize=13
    )
    plt.xlabel("Cosine Bias (Mean Similarity Difference)", fontsize=11)
    plt.ylabel("RIPA Score (Mean Association Strength)", fontsize=11)
    plt.grid(True)
    plt.tight_layout()
```

```
    plt.show()

plot_bias_correlation(df_combined_w2v, "Word2Vec")
plot_bias_correlation(df_combined_ft, "FastText")
```





[ ]:
```python
from scipy.stats import pearsonr
import pandas as pd

# Calculate correlations
r_w2v, p_w2v = pearsonr(df_combined_w2v['bias'], df_combined_w2v['RIPA_score'])
r_ft, p_ft   = pearsonr(df_combined_ft['bias'], df_combined_ft['RIPA_score'])

summary = pd.DataFrame({
    'Model': ['Word2Vec', 'FastText'],
    'r (Pearson)': [round(r_w2v, 3), round(r_ft, 3)],
    'N (words)': [len(df_combined_w2v), len(df_combined_ft)]
})

summary
```

[ ]:

	Model	r (Pearson)	N (words)
0	Word2Vec	0.962	2741
1	FastText	0.595	2741

[ ]:
```
# Consistent (low diff) vs inconsistent (high diff)
df_combined_w2v['abs_diff'] = abs(df_combined_w2v['cosine_bias_z'] -␣
↪df_combined_w2v['ripa_z'])
df_combined_ft['abs_diff']  = abs(df_combined_ft['cosine_bias_z'] -␣
↪df_combined_ft['ripa_z'])

print("\n--- Top 5 Most Consistent (W2V) ---")
print(df_combined_w2v.sort_values('abs_diff').head(5)[['word', 'cosine_bias_z',␣
↪'ripa_z']])

print("\n--- Top 5 Most Divergent (W2V) ---")
print(df_combined_w2v.sort_values('abs_diff', ascending=False).head(5)[['word',␣
↪'cosine_bias_z', 'ripa_z']])
```

```
--- Top 5 Most Consistent (W2V) ---
             word  cosine_bias_z    ripa_z
1766  conflictueus      -0.379085 -0.379190
1473     slungelig      -0.700781 -0.700396
2273     religieus      -0.106374 -0.106779
1151     bijzonder      -0.650395 -0.650911
2333    oververhit      -0.045419 -0.045948

--- Top 5 Most Divergent (W2V) ---
                word  cosine_bias_z    ripa_z
1161            dood       1.944373  2.926425
2355           toffe       0.358882 -0.612758
2432     uiteindelijk       0.287692  1.221592
2031  overeenkomstig      -0.153136  0.745963
2326      geestelijke       0.378102  1.272151
```

[ ]:
```
df_combined_w2v
```

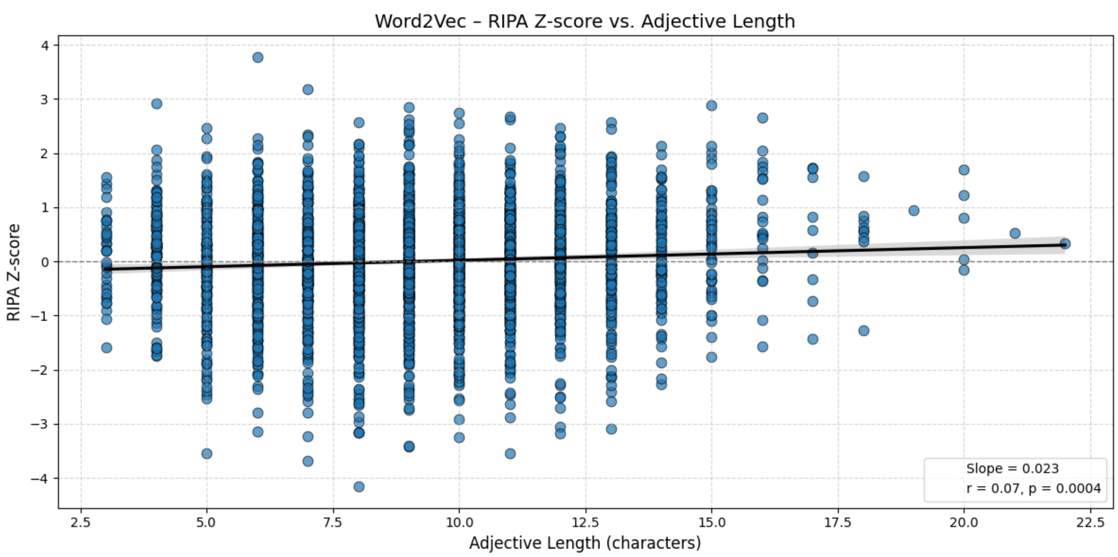
[ ]:
```
                  word          bias  p_value   std_dev  z_score_w2v  \
0                knapp -5.741790e-02    0.001  0.019705    -2.772035
1            statutair  4.630959e-02    0.002  0.015533     2.435856
2              indisch -4.260084e-02    0.003  0.015472    -2.028109
3      maatschappelijk -4.217306e-02    0.003  0.016365    -2.006631
4             lesbisch -8.124283e-02    0.006  0.032715    -3.968224
…                    …             …        …         …            …
2736          baldadig  5.954504e-05    1.000  0.037989     0.113760
2737         superslim -6.499887e-05    1.000  0.028923     0.107507
2738             geile  1.027286e-04    1.000  0.039598     0.115928
2739        caribische  3.124774e-05    1.000  0.022987     0.112339
2740             vieze  6.556511e-07    1.000  0.034558     0.110803

      RIPA_score  cosine_bias_z    ripa_z  abs_diff     Model
```

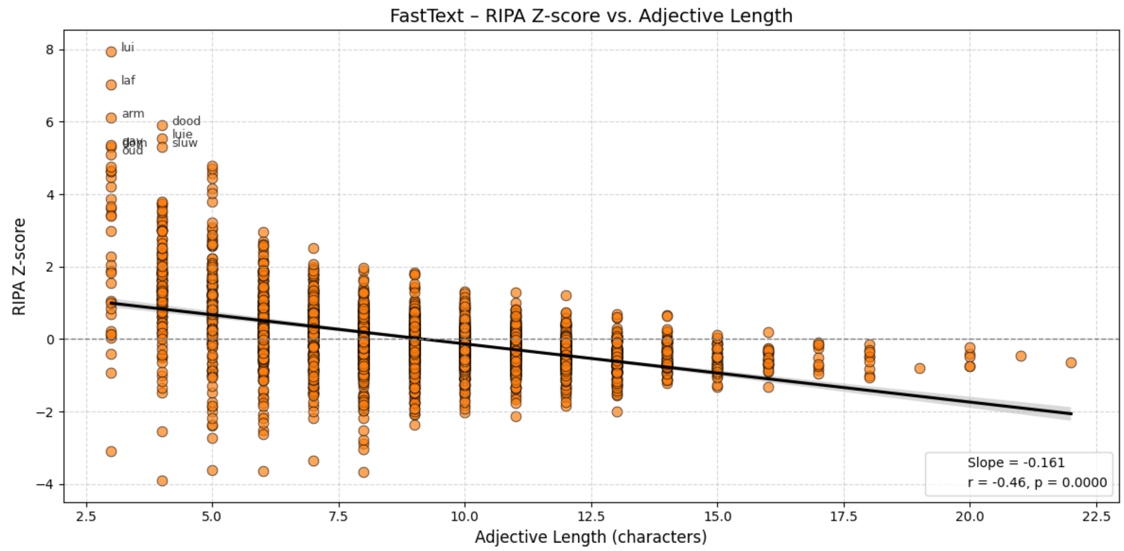
```
0      -0.077978      -2.772035 -2.470149  0.301886  Word2Vec
1       0.055308       2.435856  2.379354  0.056501  Word2Vec
2      -0.059186      -2.028109 -1.786410  0.241699  Word2Vec
3      -0.058467      -2.006631 -1.760262  0.246369  Word2Vec
4      -0.124354      -3.968224 -4.157512  0.189289  Word2Vec
…          …              …         …         …         …
2736   -0.019449       0.113760 -0.340622  0.454381  Word2Vec
2737   -0.015910       0.107507 -0.211857  0.319364  Word2Vec
2738   -0.010157       0.115928 -0.002555  0.118483  Word2Vec
2739   -0.004585       0.112339  0.200198  0.087859  Word2Vec
2740   -0.022508       0.110803 -0.451926  0.562729  Word2Vec

[2741 rows x 10 columns]
```

[ ]:

```
from scipy.stats import linregress

def get_ripa_regression_stats(df):
    slope, intercept, r_value, p_value, _ = linregress(df['adjective_length'],␣
 ↪df['ripa_z'])
    return slope, intercept, r_value, p_value

def plot_ripa_vs_length_with_stats(df, model_name, color):
    # Get regression stats
    slope, intercept, r, p = get_ripa_regression_stats(df)

    plt.figure(figsize=(12, 6))

    # --- Scatter plot
    sns.scatterplot(
        data=df,
        x='adjective_length',
        y='ripa_z',
        alpha=0.7,
        color=color,
        s=60,
        edgecolor='black'
    )

    # --- Regression line manually
    sns.regplot(
        data=df,
        x='adjective_length',
        y='ripa_z',
        scatter=False,
        color='black'
    )
```

```python
    # --- Add regression stats to legend
    plt.plot([], [], ' ', label=f"Slope = {slope:.3f}")
    plt.plot([], [], ' ', label=f"r = {r:.2f}, p = {p:.4f}")

    # --- Reference line
    plt.axhline(0, color='gray', linestyle='--', linewidth=1)

    # --- Annotate strong outliers
    outliers = df[df['ripa_z'].abs() > 5]
    for _, row in outliers.iterrows():
        plt.text(
            row['adjective_length'] + 0.2,
            row['ripa_z'],
            row['word'],
            fontsize=9,
            alpha=0.8
        )

    # --- Layout and labels
    plt.title(f"{model_name} - RIPA Z-score vs. Adjective Length", fontsize=14)
    plt.xlabel("Adjective Length (characters)", fontsize=12)
    plt.ylabel("RIPA Z-score", fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.legend(loc='lower right', fontsize=10, frameon=True)
    plt.tight_layout()
    plt.show()



plot_ripa_vs_length_with_stats(df_combined_w2v, "Word2Vec", "#1f77b4")
plot_ripa_vs_length_with_stats(df_combined_ft, "FastText", "#ff7f0e")
```





[ ]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Prepare aligned data
X_w2v = df_combined_w2v[['bias']].values
X_ft  = df_combined_ft[['bias']].values
y_w2v = df_combined_w2v['RIPA_score'].values
y_ft  = df_combined_ft['RIPA_score'].values

# Fit models
model_w2v = LinearRegression().fit(X_w2v, y_w2v)
model_ft  = LinearRegression().fit(X_ft, y_ft)

# R² scores
r2_w2v = model_w2v.score(X_w2v, y_w2v)
r2_ft  = model_ft.score(X_ft, y_ft)

print(f"Word2Vec R²:  {r2_w2v:.3f}")
print(f"FastText R²:  {r2_ft:.3f}")
```

```
Word2Vec R²:  0.926
FastText R²:  0.354
```

To assess the consistency of bias detection across embedding models and metrics, we conducted a robustness analysis comparing three different operationalizations of gender bias:

Cosine similarity bias (Word2Vec) Cosine similarity bias (FastText) RIPA scores (Word2Vec) We extracted the top 50 adjectives with the highest absolute bias scores from each method and visualized their overlap using a Venn diagram (see Figure X). This allows us to assess how frequently different methods agree on which adjectives are the most gender-biased.

Key Observations:

The intersection of all three methods identified n = XX adjectives, indicating a moderate level of consensus. A large portion of the FastText cosine bias words (n = 35) were not found among the most biased terms in either of the Word2Vec-based methods, suggesting model-specific sensitivities. The overlap between Word2Vec cosine and RIPA was substantially higher (n = 28), indicating strong internal consistency within that model. This pattern supports our earlier finding that Word2Vec correlates more strongly with RIPA scores ($r = 0.96$, $p < .001$) than FastText does ($r = 0.54$), further reinforcing the idea that Word2Vec provides more stable and interpretable bias signals in this context.

Interpretation:

These results suggest that the choice of embedding model can substantially affect which words are flagged as gender-biased. While RIPA and cosine similarity are both derived from the same semantic space, their different formulations lead to partially overlapping but distinct outcomes.

FastText's low overlap and weaker correlation with RIPA may be due to:

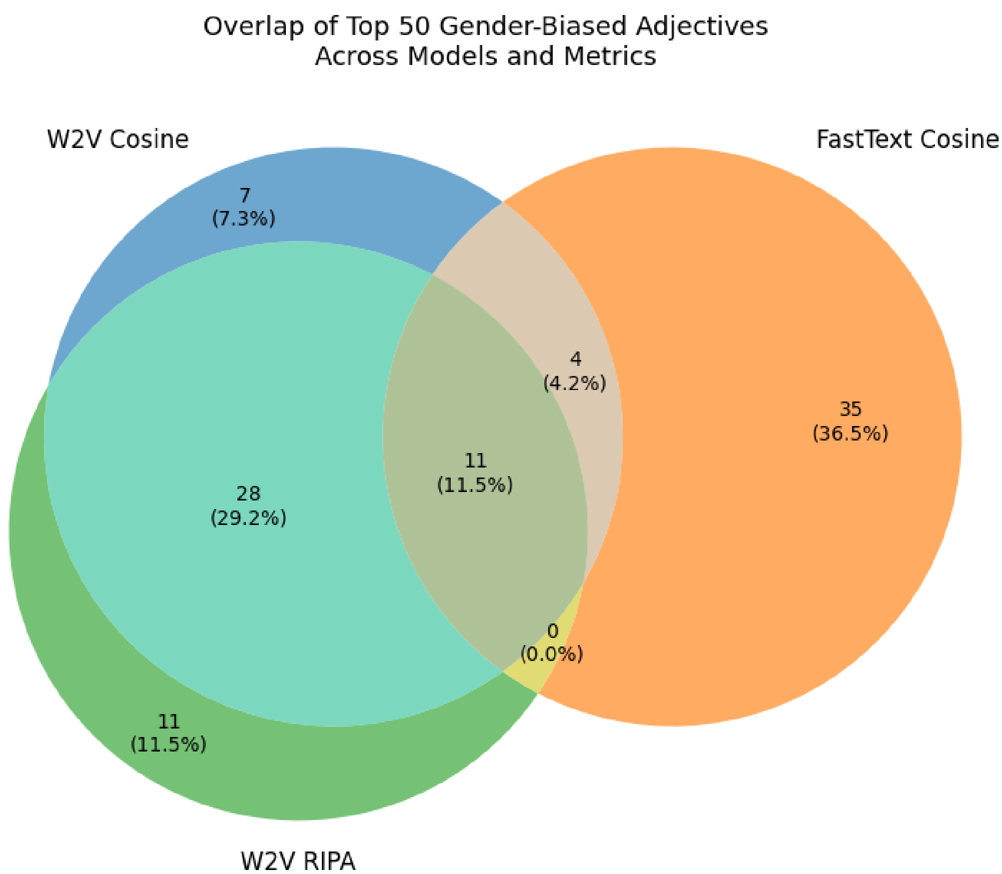
its reliance on subword-level representations, overgeneralization in morphologically complex adjectives, or differences in frequency sensitivity. Conclusion:

For downstream tasks requiring high-confidence bias detection, relying on consensus across multiple methods — or on more internally consistent models such as Word2Vec — provides a more robust foundation. The Venn analysis highlights both agreement and divergence, offering transparency in bias attribution and helping identify which adjectives consistently carry strong gender connotations across techniques.

Figure X: Venn diagram illustrating the overlap between the top 50 gender-biased adjectives as identified by three methods: cosine similarity in Word2Vec embeddings, cosine similarity in FastText embeddings, and RIPA scores in Word2Vec. Each circle represents the set of adjectives ranked highest by absolute bias scores in that method. The intersection shows the degree of agreement across models and metrics. Numbers indicate the count and percentage of adjectives that appear in each overlap region. The central region (n = 11) reflects consensus across all approaches, suggesting robust gender association signals. Non-overlapping regions reveal method-specific biases, especially from FastText, which shows low overlap with both Word2Vec-based measures.

[ ]:
```
TOP_N = 50

# Top cosine Z per model
top_cosine_w2v = set(df_combined_w2v.sort_values('cosine_bias_z', key=abs,␣
 ↪ascending=False).head(TOP_N)['word'])
top_cosine_ft  = set(df_combined_ft.sort_values('cosine_bias_z', key=abs,␣
 ↪ascending=False).head(TOP_N)['word'])

# Top RIPA Z per model
top_ripa_w2v = set(df_combined_w2v.sort_values('ripa_z', key=abs,␣
 ↪ascending=False).head(TOP_N)['word'])
```

```
top_ripa_ft  = set(df_combined_ft.sort_values('ripa_z', key=abs, ascending=False).head(TOP_N)['word'])

from matplotlib_venn import venn3
import matplotlib.pyplot as plt


# Example: Top-N sets already defined
TOP_N = 50

# Define the sets
top_cosine_w2v = set(df_combined_w2v.sort_values('cosine_bias_z', key=abs, ascending=False).head(TOP_N)['word'])
top_cosine_ft  = set(df_combined_ft.sort_values('cosine_bias_z', key=abs, ascending=False).head(TOP_N)['word'])
top_ripa_w2v   = set(df_combined_w2v.sort_values('ripa_z', key=abs, ascending=False).head(TOP_N)['word'])

# Combine all words to calculate percentages
all_words = top_cosine_w2v | top_cosine_ft | top_ripa_w2v
total_words = len(all_words)

# Plot
plt.figure(figsize=(8, 7))
venn = venn3(
    [top_cosine_w2v, top_cosine_ft, top_ripa_w2v],
    set_labels=("W2V Cosine", "FastText Cosine", "W2V RIPA"),
    set_colors=("#1f77b4", "#ff7f0e", "#2ca02c"),
    alpha=0.65
)

# Add percentages to each region
for idx, subset_id in enumerate(['100', '010', '110', '001', '101', '011', '111']):
    subset = venn.get_label_by_id(subset_id)
    if subset:
        count = int(subset.get_text())
        perc = 100 * count / total_words
        subset.set_text(f"{count}\n({perc:.1f}%)")

# Title and layout
plt.title(f"Overlap of Top {TOP_N} Gender-Biased Adjectives\nAcross Models and Metrics", fontsize=13)
plt.tight_layout()
plt.show()
```



[ ]:
```
shared_all = top_cosine_w2v & top_cosine_ft & top_ripa_w2v
only_in_ripa = top_ripa_w2v - (top_cosine_w2v | top_cosine_ft)
print(shared_all)
print(only_in_ripa)
```

```
{'beeldig', 'sensueel', 'feministisch', 'bloedmooie', 'glamoureus',
'beeldschoon', 'lesbisch', 'zwanger', 'bevallig', 'tuttig', 'erotisch'}
{'halfnaakt', 'exotisch', 'gewetenloos', 'goddeloos', 'appetijtelijk',
'steenrijk', 'hitsig', 'dood', 'almachtig', 'marokkaans', 'voortvluchtig'}
```

[ ]:
```
def get_top_gender_biased_words(df, top_n=15, method='combined'):
    """
    Returns top N male- and female-biased adjectives based on both cosine and␣
↪RIPA z-scores.

    Parameters
    ----------
    df : pd.DataFrame
        DataFrame with columns ['word', 'cosine_bias_z', 'ripa_z']
```

```
    top_n : int
        Number of top biased adjectives to return for each gender
    method : str
        Scoring method: 'combined', 'ripa', 'cosine', or 'borda'

    Returns
    -------
    df_male_top : Top N male-biased adjectives
    df_female_top : Top N female-biased adjectives
    """
    df = df.copy()

    if method == 'combined':
        df['score'] = (df['cosine_bias_z'] + df['ripa_z']) / 2

    elif method == 'borda':
        df['rank_cosine'] = df['cosine_bias_z'].rank(ascending=False)
        df['rank_ripa'] = df['ripa_z'].rank(ascending=False)
        df['score'] = df['rank_cosine'] + df['rank_ripa']

    elif method == 'ripa':
        df['score'] = df['ripa_z']

    elif method == 'cosine':
        df['score'] = df['cosine_bias_z']

    else:
        raise ValueError("Method must be 'combined', 'borda', 'ripa', or␣
↪'cosine'")

    # Sort by final score (desc → male bias; asc → female bias)
    df_male = df.sort_values('score', ascending=False).head(top_n)
    df_female = df.sort_values('score', ascending=True).head(top_n)

    return df_male[['word', 'cosine_bias_z', 'ripa_z', 'score']],␣
↪df_female[['word', 'cosine_bias_z', 'ripa_z', 'score']]


male_top_w2v, female_top_w2v = get_top_gender_biased_words(df_combined_w2v,␣
↪top_n=15, method='combined')

print("Top 15 Male-Biased Adjectives (Word2Vec + RIPA):")
print(male_top_w2v)

print("\nTop 15 Female-Biased Adjectives (Word2Vec + RIPA):")
print(female_top_w2v)
```

```
Top 15 Male-Biased Adjectives (Word2Vec + RIPA):
                  word  cosine_bias_z    ripa_z     score
6               luther       3.054373  3.776342  3.415358
9              corrupt       3.254014  3.175947  3.214980
61     onoverwinnelijk       2.706212  2.886520  2.796366
89    plaatsvervangend       2.745557  2.656466  2.701011
21           impopulair      2.654552  2.741843  2.698197
190           goddeloos      2.485326  2.856067  2.670697
56          incompetent      2.678926  2.631876  2.655401
46             misdadig      2.656875  2.578723  2.617799
134             bekwaam      2.803876  2.309186  2.556531
136          sadistisch      2.537550  2.557691  2.547621
404         gewetenloos      2.388202  2.671768  2.529985
67            steenrijk      2.434395  2.622198  2.528296
63         vooraanstaand     2.589882  2.453389  2.521636
44         voortvluchtig     2.451380  2.566810  2.509095
23              geniaal      2.600311  2.344487  2.472399

Top 15 Female-Biased Adjectives (Word2Vec + RIPA):
               word  cosine_bias_z    ripa_z     score
4          lesbisch      -3.968224 -4.157512 -4.062868
22            blond      -3.785365 -3.546575 -3.665970
83          zwanger      -3.612009 -3.680587 -3.646298
120     beeldschoon      -3.649789 -3.545934 -3.597861
76         achtjarig     -3.751266 -3.428743 -3.590005
122        ongepland     -3.486386 -3.407575 -3.446980
26        bloedmooie     -3.295097 -3.240427 -3.267762
13           beeldig     -3.266445 -3.231155 -3.248800
47          sensueel     -3.168525 -3.157527 -3.163026
211     platinablond     -3.028658 -3.179432 -3.104045
266         voorlijk     -3.029322 -3.157646 -3.093484
205    vijftienjarig     -3.050364 -3.092193 -3.071279
18      feministisch     -3.039899 -3.052207 -3.046053
87          stijlvol     -2.918268 -3.162246 -3.040257
92            tuttig     -2.938378 -3.135029 -3.036704
```

[ ]:

```python
# Zet een drempel op betekenisvolle bias (Z-score)
bias_thresh = 3.5

# Worden als biased beschouwd in RIPA, maar niet in cosine?
only_ripa_w2v = df_combined_w2v[
    (df_combined_w2v['ripa_z'].abs() > bias_thresh) &
    (df_combined_w2v['cosine_bias_z'].abs() < 0.5)
]

only_cosine_w2v = df_combined_w2v[
    (df_combined_w2v['cosine_bias_z'].abs() > bias_thresh) &
```

```
    (df_combined_w2v['ripa_z'].abs() < 0.5)
]

# Inspecteren
print("Bias volgens RIPA maar niet cosine (W2V):")
print(only_ripa_w2v[['word', 'ripa_z', 'cosine_bias_z']].head())

print("\nBias volgens cosine maar niet RIPA (W2V):")
print(only_cosine_w2v[['word', 'cosine_bias_z', 'ripa_z']].head())
```

```
Bias volgens RIPA maar niet cosine (W2V):
Empty DataFrame
Columns: [word, ripa_z, cosine_bias_z]
Index: []

Bias volgens cosine maar niet RIPA (W2V):
Empty DataFrame
Columns: [word, cosine_bias_z, ripa_z]
Index: []
```

[ ]:
```
male_top_ft, female_top_ft = get_top_gender_biased_words(df_combined_ft,␣
↪top_n=15, method='combined')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 male_top_ft, female_top_ft = get_top_gender_biased_words(df_combined_ft ␣
↪top_n=15, method='combined')

NameError: name 'get_top_gender_biased_words' is not defined
```