Method Section

Obtaining the pretrained embeddings (Google Colab):

Glove with Common Crawl (Glove with Common Crawl section of the notebook):

1. Run the following code snippet (Under the Glove Common Crawl section of the notebook):

```
!wget http://nlp.stanford.edu/data/glove.840B.300d.zip
!unzip -q glove.840B.300d.zip
```

The command in the first line (!wget) is for retrieving content from the given URL. It will result in the zip file having the glove vectors of 300 dimensions trained on the common crawl corpus get added into the current Google Colab session you are running. The file glove.840B.300d.zip will be added.

The command in the second line(!unzip) is used for unzipping the file with the given file name and returns the .txt file which contains all the tokens in the vocabulary and their vector representations which can be used for analysis.

2. Run the following code snippet (Under the Glove with Common Crawl section of the notebook)

```
embeddings_glove = {} # Empty dictionary in which the words in the
vocabulary and the corresponding vector will be added
with open("glove.840B.300d.txt") as f:
    for line in f: # Each line in the file contains the token followed by
the vector representation of that token trained in glove in 300 dimensions
separated by a space. Each dimension of the vector is separated by a space
as well
        word, coefs = line.split(maxsplit=1) # Splits the line as
described above at the first space, hence the token gets separated from
the vector. The token is stored in 'word' and the vector is stored in
'coefs'
        coefs = np.fromstring(coefs, "f", sep=" ") # Returns the vector
representation in the form of an array which can now be used for analysis
(cosine similarity, etc)
        embeddings_glove[word] = coefs # Add the token and its
corresponding vector into the dictionary as key and value pair. Now we can
call the vector of any token in the vocabulary via the dictionary
```

embeddings_glove is the dictionary in which the tokens in the vocabulary and their corresponding vector representations will be added in a key:value format.

Glove with Wikipedia and Gigaword (Glove with Wikipedia section of the notebook):

1. Run the following code snippet (Under the Glove Wikipedia section of the notebook):

```
!wget http://nlp.stanford.edu/data/glove.6B.zip # Retrieves content from the given URL
!unzip -q glove.6B.zip # Unzips the file and returns the .txt file which can be used for analysis. In this case, returns 4 files, having 50, 100, 200, and 300 dimensional vectors. We are choosing to use the 300 dimensional vectors
```

The command in the first line (!wget) is for retrieving content from the given URL. It will result in the zip file having the glove vectors of 50, 100, 200 and 300 dimensions trained on Wikipedia and Gigaword get added into the current Google Colab session you are running. The file glove.6B.zip will be added.

The command in the second line(!unzip) is used for unzipping the file with the given file name and returns the .txt file which contains all the tokens in the vocabulary and their vector representations which can be used for analysis.

2. Run the following code snippet (Under the Glove Wikipedia section of the notebook):

```
embeddings_glove_wiki_gw = {} # Empty dictionary in which the words in the vocabulary and the corresponding vector will be added
with open("glove.6B.300d.txt") as f:
    for line in f: # Each line in the file contains the token followed by a the vector representation of that token trained in glove in 300 dimensions separated by a space. Each dimension of the vector is separated by a space as well
        word, coefs = line.split(maxsplit=1) # Splits the line as described above at the first space, hence the token gets separated from the vector. The token is stored in 'word' and the vector is stored in 'coefs'
        coefs = np.fromstring(coefs, "f", sep=" ") # Returns the vector representation in the form of an array which can now be used for analysis (cosine similarity, etc)
        embeddings_glove_wiki_gw[word] = coefs # Add the token and its corresponding vector into the dictionary as key and value pair. Now we can call the vector of any token in the vocabulary via the dictionary
```

embeddings_glove_wiki is the dictionary in which the tokens in the vocabulary and their corresponding vector representations will be added in a key:value format.

Fastext with Common Crawl (Fastext Common Crawl section of the notebook):

1. Run the following code snippet (Under the Fastext Common Crawl section of the notebook):

```
!wget
https://dl.fbaipublicfiles.com/fasttext/vectors-english/crawl-300d-2M.vec.
zip # Retrieves content from the given URL
!unzip -q crawl-300d-2M.vec.zip # Unzips the file and returns the .vec
file which can be used for analysis
```

The command in the first line (!wget) is for retrieving content from the given URL. It will result in the zip file having the glove vectors of 300 dimensions trained on Common Crawl get added into the current Google Colab session you are running. The file crawl-300d-2M.vec.zip will be added.

The command in the second line(!unzip) is used for unzipping the file with the given file name and returns the .vec file which contains all the tokens in the vocabulary and their vector representations which can be used for analysis.

2. Run the following code snippet (Under the Fastext Common Crawl section of the notebook):

```
embeddings_fastext_cc = {}
with open("crawl-300d-2M.vec") as f:
    for line in f: # Each line in the file contains the token followed by
a the vector representation of that token trained in glove in 300
dimensions separated by a space. Each dimension of the vector is separated
by a space as well
        word, coefs = line.split(maxsplit=1) # Splits the line as
described above at the first space, hence the token gets separated from
the vector. The token is stored in 'word' and the vector is stored in
'coefs'
        coefs = np.fromstring(coefs, "f", sep=" ") # Returns the vector
representation in the form of an array which can now be used for analysis
(cosine similarity, etc)
        embeddings_fastext_cc[word] = coefs # Add the token and its
corresponding vector into the dictionary as key and value pair. Now we can
call the vector of any token in the vocabulary via the dictionary
```

embeddings_fastext_cc is the dictionary in which the tokens in the vocabulary and their corresponding vector representations will be added in a key:value format

.Fastext with Wikipedia, UBMC, statmt.org (Fastext Wikipedia section of the notebook):

1. Run the following code snippet (Under the Fastext Wikipedia section of the notebook):

```
!wget
https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M.
vec.zip # Retrieves content from the given URL
!unzip -q wiki-news-300d-1M.vec.zip # Unzips the file and returns the .vec
file which can be used for analysis
```

The command in the first line (!wget) is for retrieving content from the given URL. It will result in the zip file having the glove vectors of 300 dimensions trained on Wikipedia, UBMC, statmt.org get added into the current Google Colab session you are running. The file wiki-news-300d-1M.vec.zip will be added.

The command in the second line(!unzip) is used for unzipping the file with the given file name and returns the .vec file which contains all the tokens in the vocabulary and their vector representations which can be used for analysis.

2. Run the following code snippet (Under the Fastext Wikipedia section of the notebook):

```
embeddings_fastext_wiki = {}
with open("wiki-news-300d-1M.vec") as f:
    for line in f: # Each line in the file contains the token followed by
a the vector representation of that token trained in glove in 300
dimensions separated by a space. Each dimension of the vector is separated
by a space as well
        word, coefs = line.split(maxsplit=1) # Splits the line as
described above at the first space, hence the token gets separated from
the vector. The token is stored in 'word' and the vector is stored in
'coefs'
        coefs = np.fromstring(coefs, "f", sep=" ") # Returns the vector
representation in the form of an array which can now be used for analysis
(cosine similarity, etc)
```

```
        embeddings_fastext_cc[word] = coefs # Add the token and its
corresponding vector into the dictionary as key and value pair. Now we can
call the vector of any token in the vocabulary via the dictionary
```

embeddings_fastext_wiki is the dictionary in which the tokens in the vocabulary and their corresponding vector representations will be added in a key:value format.

Word2vec with Google News (word2vec section of the notebook):

1. Run the following code snippet (under the word2vec section of the notebook):

```
!wget -P /root/input/ -c
"https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300
.bin.gz"
```

The command in the first line (!wget) is for retrieving content from the given URL. It will result in the zip file having the word2vec of 300 dimensions trained on GoogleNews get added into the current Google Colab session you are running. The file GoogleNews-vectors-negative300.bin.gz will be added.

2. Run the following code snippet (under the word2vec section of the notebook):

```
EMBEDDING_FILE = '/root/input/GoogleNews-vectors-negative300.bin.gz'
embeddings_word2vec = KeyedVectors.load_word2vec_format(EMBEDDING_FILE,
binary=True)
```

embeddings_word2vec is the dictionary in which the tokens in the vocabulary and their corresponding vector representations will be added in a key:value format.

Calculating the WEAT Effect sizes and p-values (Under the WEAT section of the notebook):

The function which returns the WEAT Effect Size is weat_ES. The function which returns the p-value is p_value_weat.

Getting the WEAT Effect Size:

For getting the WEAT ES, the function ‘weat_ES’ needs to be run. The function called within this function is ‘weat_swAB’ which in turn calls the ‘cos’ function. Hence the functions ‘cos’ and ‘weat_swAB’ need to be run in chronological order before the ‘weat_ES’ function is run.

1. cos function: Returns the cosine similarity of 2 word vectors
2. weat_sWAB function: Returns the association of a single word with the attribute words (eg. the association of the word ‘Alcohol’ with the attribute words for ‘good’ and ‘bad’)
3. **weat_ES function**: Returns the effect size of a pair of categories (eg Simple vs Difficult with the chosen attribute (eg. good/bad).

Check the notebook for details about the arguments of each function.

Getting the p-value:
For getting the WEAT p-value, the function ‘p_value_weat’ needs to be run. The functions called within this function are ‘random_permutation’ and ‘test_statistic’. These functions need to be run before the ‘p_value_weat’ function is run.

1. test_statistic function: Returns the test statistic given the 2 sets of category stimuli and attribute stimuli. Defined in Caliskan et al., 2017.
2. random_permutation: Returns a random permutation of a fixed number of a chosen list
3. **p_value_weat:** Returns the p-value given a pair of categories (eg Simple vs Difficult) and the chosen attribute (eg. good/bad)

Check the notebook for details about the arguments for each function.