In [10]:
```
import pandas as pd
import numpy as np
import nltk
import re
import string

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from gensim.models import Word2Vec
import matplotlib.pyplot as plt
import seaborn as sns
```

In [11]:
```
df = pd.read_csv('data.csv')

print("Shape:", df.shape)
print("\nColumns:", df.columns.tolist())
print("\nFirst 3 rows:")
df.head(3)
```

```
Shape: (11914, 16)

Columns: ['Make', 'Model', 'Year', 'Engine Fuel Type', 'Engine HP', 'Engine Cylinders', 'Transmission Type', 'Driven_
Wheels', 'Number of Doors', 'Market Category', 'Vehicle Size', 'Vehicle Style', 'highway MPG', 'city mpg', 'Popularit
y', 'MSRP']

First 3 rows:
```

Out[11]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	V
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Conv
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	

In [12]:
```
text_columns = ['Make', 'Model', 'Engine Fuel Type', 'Transmission Type',
                'Driven_Wheels', 'Market Category', 'Vehicle Size', 'Vehicle Style']

# Fill NaN and combine columns into one text field
df['text'] = df[text_columns].fillna('').astype(str).agg(' '.join, axis=1)

print("Sample text entries:")
for i, txt in enumerate(df['text'].head(3)):
    print(f"\n[{i}] {txt}")
```

```
Sample text entries:

[0] BMW 1 Series M premium unleaded (required) MANUAL rear wheel drive Factory Tuner,Luxury,High-Performance Compact
Coupe

[1] BMW 1 Series premium unleaded (required) MANUAL rear wheel drive Luxury,Performance Compact Convertible

[2] BMW 1 Series premium unleaded (required) MANUAL rear wheel drive Luxury,High-Performance Compact Coupe
```

In [13]:
```
# Cell 4: Preprocess the text corpus

def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)        # remove punctuation/digits
    text = re.sub(r'\s+', ' ', text).strip()     # collapse whitespace
    return text

df['clean_text'] = df['text'].apply(preprocess)

print("Sample cleaned text:")
```

```
for i, txt in enumerate(df['clean_text'].head(3)):
    print(f"\n[{i}] {txt}")
```

```
Sample cleaned text:

[0] bmw series m premium unleaded required manual rear wheel drive factory tunerluxuryhighperformance compact coupe

[1] bmw series premium unleaded required manual rear wheel drive luxuryperformance compact convertible

[2] bmw series premium unleaded required manual rear wheel drive luxuryhighperformance compact coupe
```

In [14]:
```
# Cell 5: Bag of Words – Raw Count Occurrences

count_vectorizer = CountVectorizer(max_features=50, stop_words='english')
bow_matrix = count_vectorizer.fit_transform(df['clean_text'])

bow_df = pd.DataFrame(bow_matrix.toarray(),
                      columns=count_vectorizer.get_feature_names_out())

print("BoW Matrix Shape:", bow_df.shape)
print("\nSample (first 5 rows, first 10 features):")
bow_df.iloc[:5, :10]
```

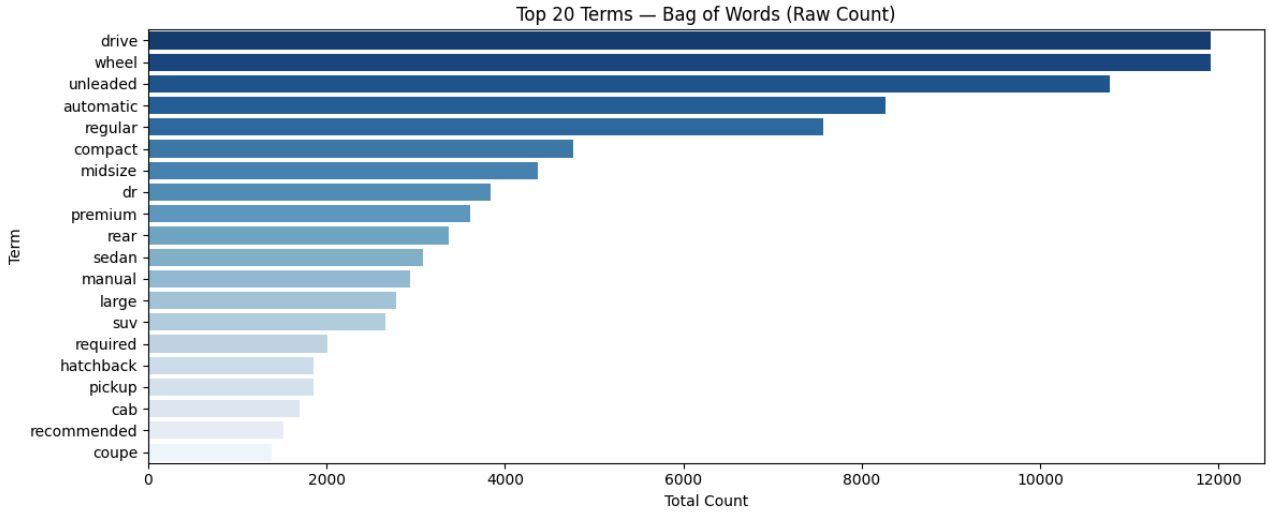
```
BoW Matrix Shape: (11914, 50)

Sample (first 5 rows, first 10 features):
```

Out[14]:

	automatedmanual	automatic	bmw	cab	cadillac	chevrolet	compact	convertible	coupe	crew
0	0	0	1	0	0	0	1	0	1	0
1	0	0	1	0	0	0	1	1	0	0
2	0	0	1	0	0	0	1	0	1	0
3	0	0	1	0	0	0	1	0	1	0
4	0	0	1	0	0	0	1	1	0	0

In [15]:
```
# Cell 6: Visualize top 20 most frequent terms (BoW)

word_freq = bow_df.sum().sort_values(ascending=False).head(20)

plt.figure(figsize=(12, 5))
sns.barplot(x=word_freq.values, y=word_freq.index, palette='Blues_r')
plt.title('Top 20 Terms – Bag of Words (Raw Count)')
plt.xlabel('Total Count')
plt.ylabel('Term')
plt.tight_layout()
plt.show()
```

```
C:\Users\Lalit H\AppData\Local\Temp\ipykernel_5720\3693083977.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `h
ue` and set `legend=False` for the same effect.

  sns.barplot(x=word_freq.values, y=word_freq.index, palette='Blues_r')
```



In [16]:
```
# Cell 7: Normalized BoW (relative term frequency per document)
# Divide each row by its total word count so rows sum to 1

bow_array = bow_matrix.toarray().astype(float)
row_sums = bow_array.sum(axis=1, keepdims=True)
row_sums[row_sums == 0] = 1   # avoid division by zero

normalized_bow = bow_array / row_sums

norm_bow_df = pd.DataFrame(normalized_bow,
                           columns=count_vectorizer.get_feature_names_out())

print("Normalized BoW Matrix Shape:", norm_bow_df.shape)
print("\nRow sum check (should all be ~1.0):")
print(norm_bow_df.sum(axis=1).head())

print("\nSample normalized values (first 5 rows, first 10 features):")
norm_bow_df.iloc[:5, :10].round(4)
```

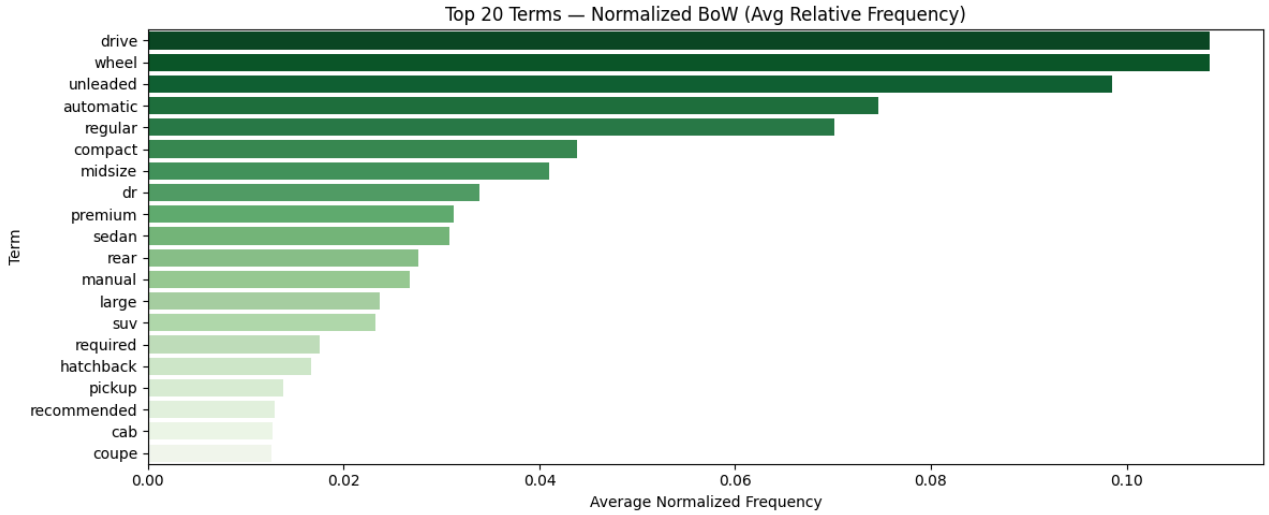
```
Normalized BoW Matrix Shape: (11914, 50)

Row sum check (should all be ~1.0):
0    1.0
1    1.0
2    1.0
3    1.0
4    1.0
dtype: float64

Sample normalized values (first 5 rows, first 10 features):
```

Out[16]:

	automatedmanual	automatic	bmw	cab	cadillac	chevrolet	compact	convertible	coupe	crew
0	0.0	0.0	0.0909	0.0	0.0	0.0	0.0909	0.0000	0.0909	0.0
1	0.0	0.0	0.0909	0.0	0.0	0.0	0.0909	0.0909	0.0000	0.0
2	0.0	0.0	0.1000	0.0	0.0	0.0	0.1000	0.0000	0.1000	0.0
3	0.0	0.0	0.0909	0.0	0.0	0.0	0.0909	0.0000	0.0909	0.0
4	0.0	0.0	0.0909	0.0	0.0	0.0	0.0909	0.0909	0.0000	0.0

In [17]:
```
# Cell 8: Visualize average normalized frequency for top 20 terms

avg_norm = norm_bow_df.mean().sort_values(ascending=False).head(20)

plt.figure(figsize=(12, 5))
sns.barplot(x=avg_norm.values, y=avg_norm.index, palette='Greens_r')
plt.title('Top 20 Terms — Normalized BoW (Avg Relative Frequency)')
plt.xlabel('Average Normalized Frequency')
plt.ylabel('Term')
plt.tight_layout()
plt.show()
```

```
C:\Users\Lalit H\AppData\Local\Temp\ipykernel_5720\2388360706.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `h
ue` and set `legend=False` for the same effect.

  sns.barplot(x=avg_norm.values, y=avg_norm.index, palette='Greens_r')
```



In [18]:
```
# Cell 9: TF-IDF (Term Frequency - Inverse Document Frequency)

tfidf_vectorizer = TfidfVectorizer(max_features=50, stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(df['clean_text'])

tfidf_df = pd.DataFrame(tfidf_matrix.toarray(),
                        columns=tfidf_vectorizer.get_feature_names_out())

print("TF-IDF Matrix Shape:", tfidf_df.shape)
print("\nSample TF-IDF values (first 5 rows, first 10 features):")
tfidf_df.iloc[:5, :10].round(4)
```

```
TF-IDF Matrix Shape: (11914, 50)

Sample TF-IDF values (first 5 rows, first 10 features):
```

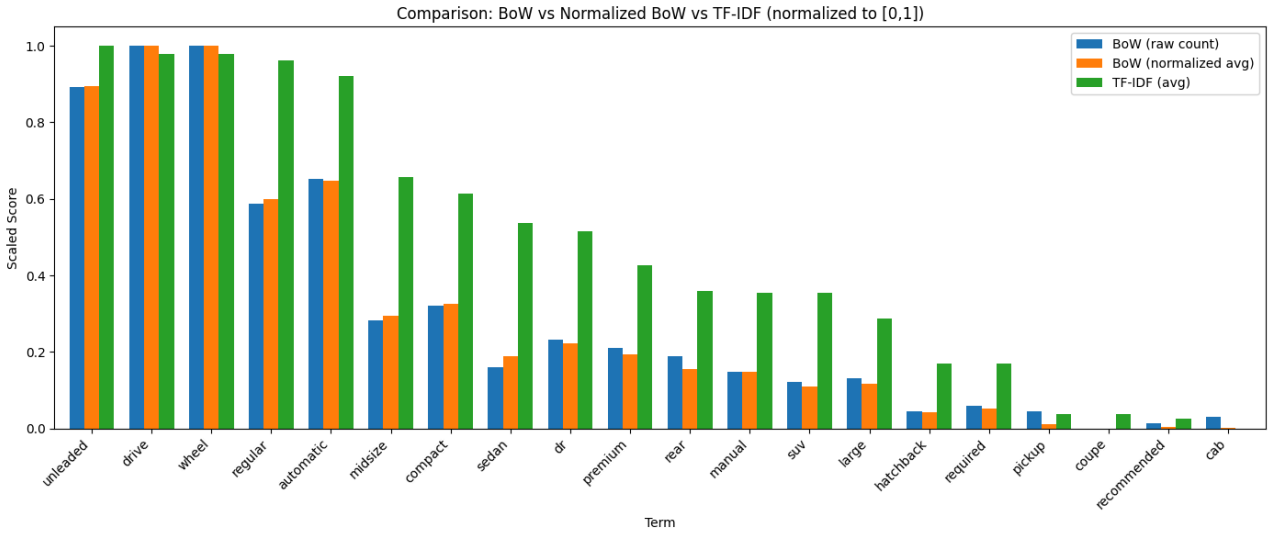
Out[18]:

	automatedmanual	automatic	bmw	cab	cadillac	chevrolet	compact	convertible	coupe	crew
0	0.0	0.0	0.5107	0.0	0.0	0.0	0.2141	0.0000	0.3633	0.0
1	0.0	0.0	0.5121	0.0	0.0	0.0	0.2147	0.4114	0.0000	0.0
2	0.0	0.0	0.5816	0.0	0.0	0.0	0.2438	0.0000	0.4138	0.0
3	0.0	0.0	0.5217	0.0	0.0	0.0	0.2187	0.0000	0.3712	0.0
4	0.0	0.0	0.5180	0.0	0.0	0.0	0.2172	0.4161	0.0000	0.0

In [19]:
```
avg_tfidf = tfidf_df.mean().sort_values(ascending=False).head(20)
```

In [20]:
```
# Cell 11: Side-by-side comparison of all three approaches

top_terms = avg_tfidf.index.tolist()   # use TF-IDF top terms as baseline

comparison_df = pd.DataFrame({
    'BoW (raw count)':        bow_df[top_terms].sum(),
    'BoW (normalized avg)':   norm_bow_df[top_terms].mean(),
    'TF-IDF (avg)':           tfidf_df[top_terms].mean()
})

# Normalize each column 0-1 for fair visual comparison
comparison_norm = (comparison_df - comparison_df.min()) / \
                  (comparison_df.max() - comparison_df.min())

fig, ax = plt.subplots(figsize=(14, 6))
comparison_norm.plot(kind='bar', ax=ax, width=0.75)
ax.set_title('Comparison: BoW vs Normalized BoW vs TF-IDF (normalized to [0,1])')
```

```
ax.set_xlabel('Term')
ax.set_ylabel('Scaled Score')
ax.legend(loc='upper right')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [21]:
```
tokenized_corpus = [text.split() for text in df['clean_text']]

print(f"Total documents: {len(tokenized_corpus)}")
print(f"Sample tokens from doc[0]: {tokenized_corpus[0][:15]}")
```

```
Total documents: 11914
Sample tokens from doc[0]: ['bmw', 'series', 'm', 'premium', 'unleaded', 'required', 'manual', 'rear', 'wheel', 'driv
e', 'factory', 'tunerluxuryhighperformance', 'compact', 'coupe']
```

In [22]:
```
# Cell 13: Train Word2Vec model

w2v_model = Word2Vec(
    sentences=tokenized_corpus,
    vector_size=100,    # embedding dimensions
    window=5,           # context window size
    min_count=2,        # ignore words that appear less than 2 times
    workers=4,          # parallel threads
    sg=1,               # 1 = Skip-gram, 0 = CBOW
    epochs=10,
    seed=42
)

vocab_size = len(w2v_model.wv)
print(f"Vocabulary size: {vocab_size}")
print(f"Embedding dimensions: {w2v_model.wv.vector_size}")
print(f"\nSample vocabulary: {list(w2v_model.wv.key_to_index.keys())[:20]}")
```

```
Vocabulary size: 723
Embedding dimensions: 100

Sample vocabulary: ['drive', 'wheel', 'unleaded', 'automatic', 'regular', 'front', 'compact', 'midsize', 'dr', 'premi
um', 'rear', 'sedan', 'manual', 'large', 'suv', 'all', 'required', 'hatchback', 'pickup', 'cab']
```

In [23]:
```
# Cell 14: Explore Word2Vec embeddings — similar words

test_words = ['toyota', 'luxury', 'automatic', 'suv', 'diesel']

for word in test_words:
    if word in w2v_model.wv:
        similar = w2v_model.wv.most_similar(word, topn=5)
        print(f"\nTop 5 words similar to '{word}':")
        for w, score in similar:
            print(f"  {w:<20} similarity: {score:.4f}")
    else:
        print(f"\n'{word}' not in vocabulary")
```

```
Top 5 words similar to 'toyota':
  solara               similarity: 0.5618
  camry                similarity: 0.5307
  prius                similarity: 0.5292
  hatchbackhybrid      similarity: 0.5037
  kia                  similarity: 0.4784

Top 5 words similar to 'luxury':
  luxuryperformance    similarity: 0.8727
  luxuryhighperformance similarity: 0.7745
  crossoverluxury      similarity: 0.7729
  luxuryhighperformancehybrid similarity: 0.7280
  h                    similarity: 0.6980

Top 5 words similar to 'automatic':
  xuv                  similarity: 0.4676
  hd                   similarity: 0.4570
  envision             similarity: 0.4419
  custom               similarity: 0.4399
  rendezvous           similarity: 0.4376

Top 5 words similar to 'suv':
  cx                   similarity: 0.6062
  nitro                similarity: 0.6052
  dr                   similarity: 0.5813
  mkx                  similarity: 0.5785
  xl                   similarity: 0.5659

Top 5 words similar to 'diesel':
  dieselluxury         similarity: 0.7920
  crossoverluxurydiesel similarity: 0.6934
  sportwagen           similarity: 0.6496
  crossoverdiesel      similarity: 0.6198
  hatchbackdiesel      similarity: 0.6032
```

In [24]:

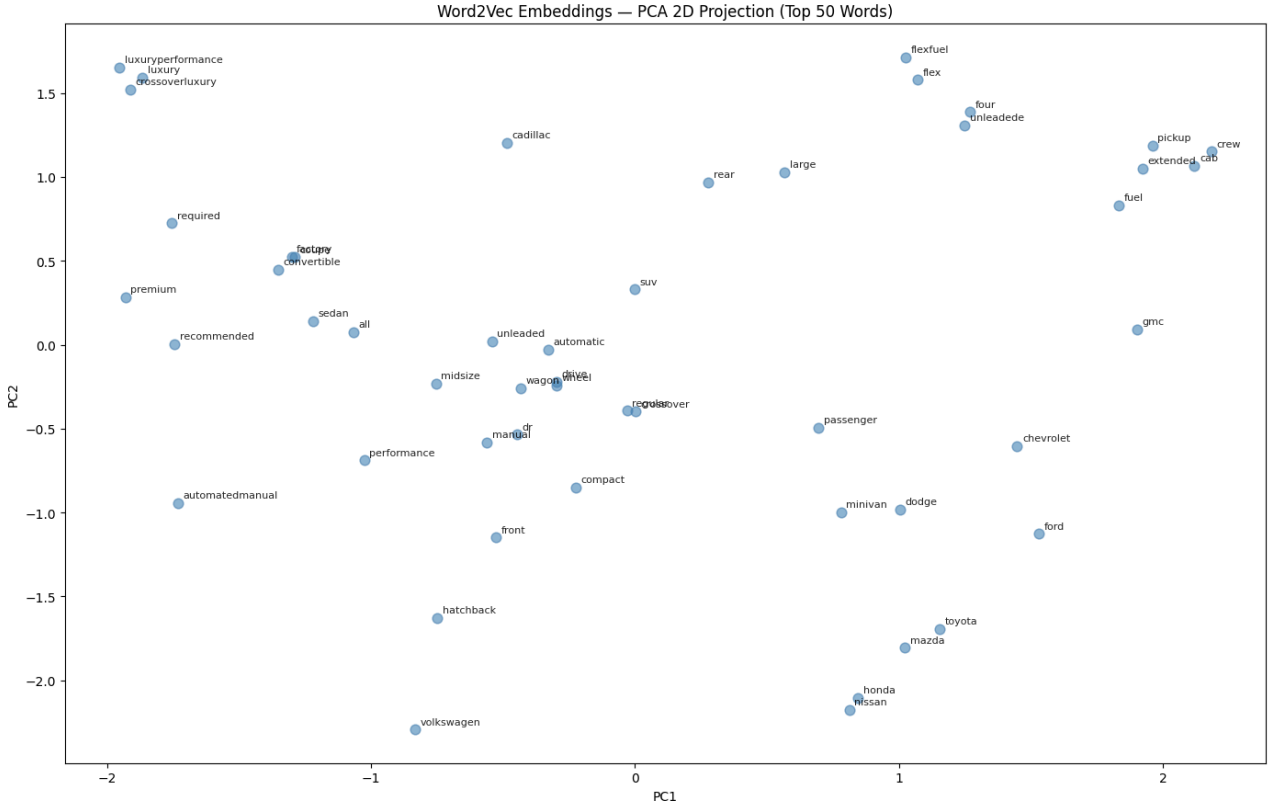
```python
# Cell 15: Create document-level embeddings by averaging word vectors

def doc_embedding(tokens, model):
    vectors = [model.wv[w] for w in tokens if w in model.wv]
    if vectors:
        return np.mean(vectors, axis=0)
    return np.zeros(model.wv.vector_size)

doc_embeddings = np.array([doc_embedding(tokens, w2v_model)
                           for tokens in tokenized_corpus])

print(f"Document embeddings shape: {doc_embeddings.shape}")
print(f"\nEmbedding for doc[0] (first 10 dims):\n{doc_embeddings[0][:10].round(4)}")
```

```
Document embeddings shape: (11914, 100)

Embedding for doc[0] (first 10 dims):
[-0.0476  0.4716 -0.1184 -0.0289 -0.1434  0.1013 -0.2683  0.1909 -0.2833
 -0.3727]
```

In [25]:

```python
# Cell 16: Visualize Word2Vec embeddings using PCA (2D)

from sklearn.decomposition import PCA

# Pick top 50 most frequent vocabulary words to plot
top_words = [w for w, _ in w2v_model.wv.key_to_index.items()][:50]
word_vectors = np.array([w2v_model.wv[w] for w in top_words])

pca = PCA(n_components=2, random_state=42)
reduced = pca.fit_transform(word_vectors)

plt.figure(figsize=(14, 9))
plt.scatter(reduced[:, 0], reduced[:, 1], alpha=0.6, s=60, color='steelblue')

for i, word in enumerate(top_words):
    plt.annotate(word, (reduced[i, 0], reduced[i, 1]),
                 fontsize=8, alpha=0.85,
                 xytext=(4, 4), textcoords='offset points')

plt.title('Word2Vec Embeddings — PCA 2D Projection (Top 50 Words)')
```

```
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.tight_layout()
plt.show()
```



In [27]:
```
print(f"\n{'Approach':<35} {'Shape'}")
print("-" * 55)
print(f"{'Bag of Words (raw count)':<35} {str(bow_df.shape)}")
print(f"{'Bag of Words (normalized)':<35} {str(norm_bow_df.shape)}")
print(f"{'TF-IDF':<35} {str(tfidf_df.shape)}")
print(f"{'Word2Vec (doc embeddings)':<35} {str(doc_embeddings.shape)}")

print("\n--- Key Parameters ---")
print(f"Vocabulary (BoW/TF-IDF): top 50 features")
print(f"Word2Vec vector size   : {w2v_model.wv.vector_size}")
print(f"Word2Vec vocab size    : {len(w2v_model.wv)}")
print(f"Total documents        : {len(df)}")
print("=" * 55)
```

```
Approach                            Shape
-------------------------------------------------------
Bag of Words (raw count)            (11914, 50)
Bag of Words (normalized)           (11914, 50)
TF-IDF                              (11914, 50)
Word2Vec (doc embeddings)           (11914, 100)

--- Key Parameters ---
Vocabulary (BoW/TF-IDF): top 50 features
Word2Vec vector size   : 100
Word2Vec vocab size    : 723
Total documents        : 11914
=======================================================
```

In [ ]: