

**T.C**

**GÜMÜŞHANE ÜNİVERSİTESİ**

**MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKULTESİ**

**YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ**

**Moda Kombin Uyumu Analiz Projesi**

**GÜLŞEN ÇİNTUĞLU**

**2107231046**

Metin Tabanlı Veri Seti Üzerinde Analiz: Moda Kombin Uyumu Analiz Projesi

**1.GİRİŞ**

Bu ödevin amacı, metin tabanlı bir veri seti üzerinden doğal dil işleme yöntemleri uygulayarak farklı vektörleştirme teknikleri ile analizler yapmaktır. Örnek çalışma olarak Zara firmasına ait ürün açıklama metinleri kullanılmış, ön işleme, Zipf analizi, TF-IDF ve Word2Vec yöntemleri uygulanmıştır. Çalışmanın sonunda bu tekniklerin performansları karşılaştırılmış ve kelime dağılımları üzerine yorumlar getirilmiştir.

Not: Çalışmanın ilk aşamasında, veri setinde yer alan açıklama metinleri İngilizce'den Türkçe'ye çevrilmiş ve Türkçe doğal dil işleme analizleri için Zemberek kütüphanesi kullanılmaya çalışılmıştır. Ancak çeviri sonrası anlam kayıpları, dil modeli desteği ve kütüphane sınırlılıkları nedeniyle yeterli başarı elde edilememiştir. Bu sebeple çalışma, orijinal İngilizce veri seti üzerinden sürdürülmüştür.

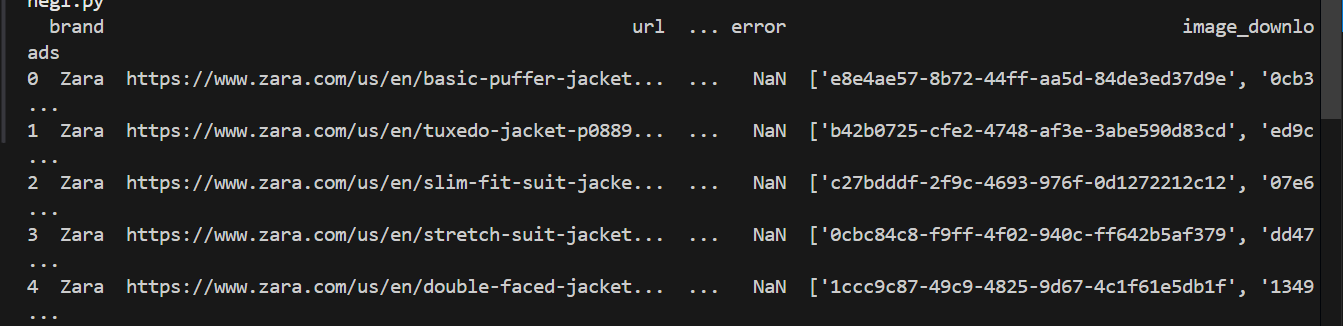
**2.HAM VERİ SETİ BİLGİLERİ**

Kaynak: Kaggle üzerinden temin edilen "Zara Sales Products Analysis" veri seti kullanılmıştır. İlgili bağlantı:

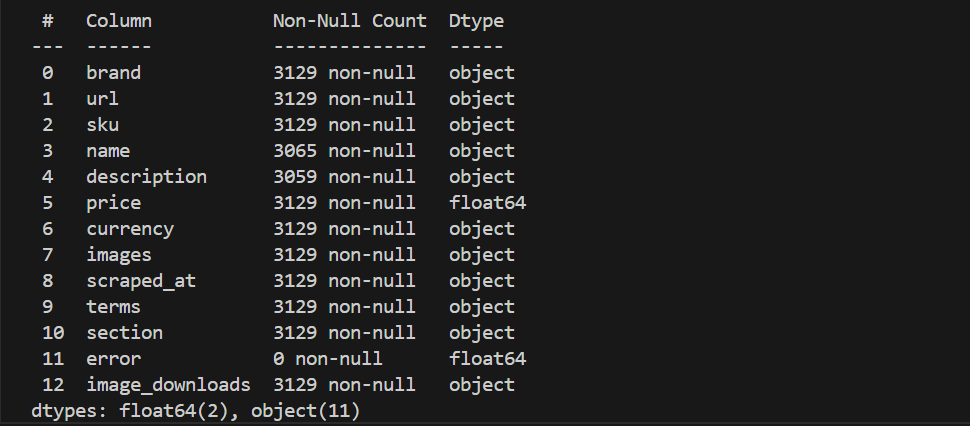
<https://www.kaggle.com/code/nadiraahmadi/zara-sales-products-analysis?select=store_zara.csv>

CSV formatında 3129 satır, 13 sütundan oluşan veri seti, bu proje kapsamında yalnızca “name”, “description” ve “section” sütunları ile sınırlandırılarak öznitelik sayısı düşürülmüştür. Veri setinde eksik veriler kontrol edilmiş ve ön işleme adımları uygulanmıştır.

Örnekler:



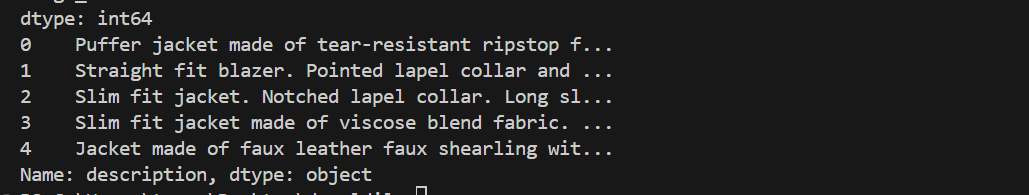
Şekil 1. Veri Seti İlk Beş Satır



Şekil 2. Veri Setide Bulunan Öznitelikler



Şekil 3. Veri Setinde Eksik Veri Kontrolü



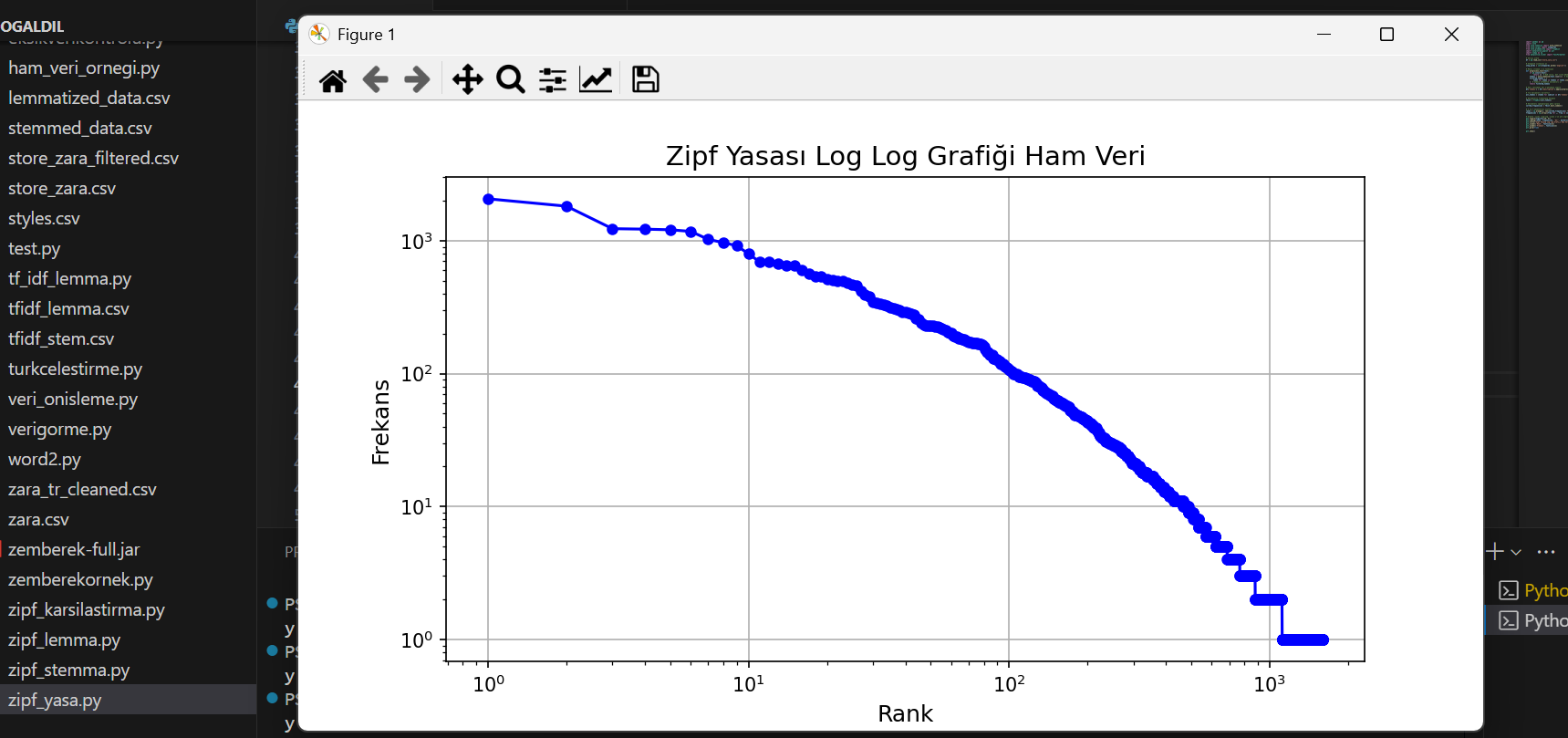
Şekil 4. Description Sütununa Genel Bakış

**Şekil 1**, ham veri setinin ilk beş satırının yazdırılmasıyla elde edilen çıktıyı göstermektedir.  
**Şekil 2**'de, veri temizleme işlemleri uygulanmadan önce veri setinde yer alan tüm sütunlar sunulmuştur.  
**Şekil 3**, her bir sütuna karşılık gelen eksik veri sayısını görsel olarak ifade etmektedir.  
Son olarak, **Şekil 4**'te açıklama sütununun örnek içerikleri gösterilmektedir.

Veri Setini tanıma aşamasında kullandığım kodlara ulaşmak için Github bağlantımı ziyaret edebilirsiniz: [homework/src/ham\_veri\_ornegi.py at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/src/ham_veri_ornegi.py)

**3.ZİPF YASASI ANALİZİ**

Ham veri üzerinden oluşturulan log-log grafiği, kelime frekanslarının Zipf dağılımına uygun olduğunu göstermiştir. Ham veri için çizilen log log grafiği **Şekil 5’**te görüldüğü üzeredir.



Şekil 5. Ham Veri için Log Log Grafiği

Seçilen veri setinin boyut olarak analiz için uygunluğu log log grafiğinde gözlemlenen düzenli dağılımdan anlaşılmaktadır. Grafikte sol üstten sağ alta doğru azalan bir eğilim izlenmesi, veri setinin Zipf Yasası’na uygun bir yapı sergilediğini göstermektedir. Toplamda 52.256 kelimelik bir kapsamla, sık kullanılan terimlerden nadiren geçen ifadelere kadar geniş bir frekans aralığı elde edilmiştir.  
**Şekil 5**’te görüldüğü üzere, yüksek frekansa sahip az sayıdaki kelime ile düşük frekansta çok sayıda kelimenin varlığı, bu dağılımın Zipf Yasası’na uygun olduğunu doğrulamaktadır.

İlgili analizlerde yazılan kodlara bu bağlantı üzerinden ulaşabilirsiniz:

<https://github.com/tucecnt/homework/blob/main/src/zipf_yasa_hamveri.py>

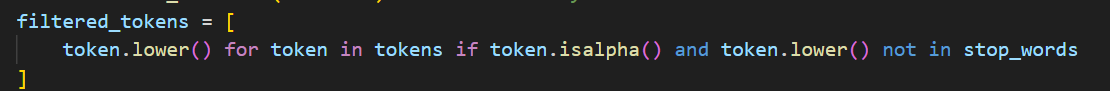
**4. ÖN İŞLEME AŞAMALARI**

* 1. **Stop Word Removal**

Bu adımda kullanılan kütüphane “Nltk.corpus.stopwords” kütüphanesidir. stop\_words listesi oluşturup anlam katmayan kelimeler bu değişkende tutuluyor.



Şekil 6. Stop Word İşlemleri-1



Şekil 7. Stop Word İşlemleri-2

Şekil 7 de görülen “Filtered\_tokens” listesinde ise tutmuş olduğum stop\_words listesindeki kelimeler çıkartılıyor.

Örnek olarak Şekil 8 verilebilir. Görüldüğü üzere “of” kelimesi cümleden çıkarılmıştır.



Şekil 8. Stop Word Örneği

* 1. **Tokenization**

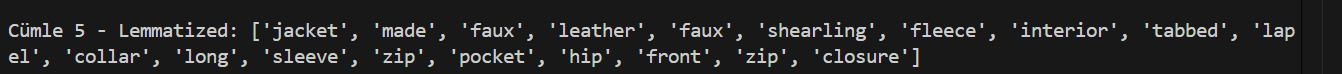
Bu aşamada metni kelimelere ve cümlelere ayırma işlemini gerçekleştirdim. Kullandığım kütüphane ise “nltk.tokenize” kütüphanesidir. Cümlelere ayırma işlemi için sent\_tokenize, kelimelere ayırma işlemi için ise Word\_tokenize kullandım.



Şekil 9. Cümlelere Ayırma İşlemi



Şekil 10. Kelimelere Ayırma İşlemi



Şekil 11. Tokenization Örneği

**Şekil 11** de görüldüğü üzere cümlelere ve kelimelere ayırma örneği gösterilmektedir.

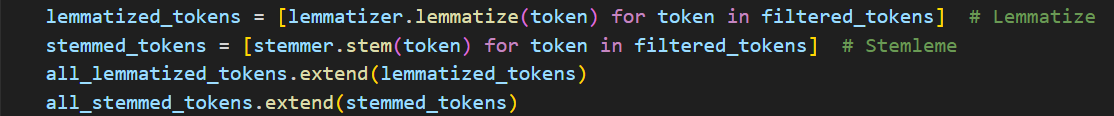
* 1. **Lowercasing**

Pythonda yerleşik olarak bulunan lower fonksiyonu ile bu işlemi gerçekleştirdim. Şekil-7 de gösterilen şekilde kullanımını gerçekleştirdim. Böylece tüm harfleri küçük olarak aldım.

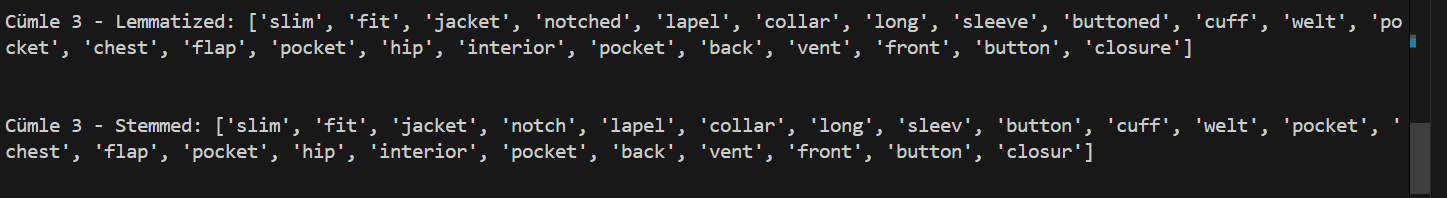
* 1. **Lemmatization ve Stemming**

Kelimelerin kök halini bulmak için NLTK kütüphanesine ait “WordNetLemmatizer” kullandım. Kelimenin anlamını koruyarak kök haline getirilmesini sağlamak için bu işlemi gerçekleştirdim.

Kelimelerin anlamlı veya anlamsız kök halini bulmak için ise NLTK kütüphanesine ait “PorterStemmer” kullandım. **Şekil 12** de görüldüğü üzere iki adımı da gerçekleştirdim.



Şekil 12. Lemma ve Stem Kodları



Şekil 13. Örnek Çıktılar

**Şekil 13**’te görüldüğü üzere aynı cümle için hem lemma hem stem işlemleri gösterilmiştir. Örneğin “notched” kelimesini lemma aynı şekilde alırken stem “notch” olarak almıştır.

Son olarak veri işlemeyi tamamlayarak iki ayrı csv dosyası olarak kaydettim. İlgili kodların hepsine bu linkten ulaşabilirsiniz: <https://github.com/tucecnt/homework/blob/main/src/data_process.py>

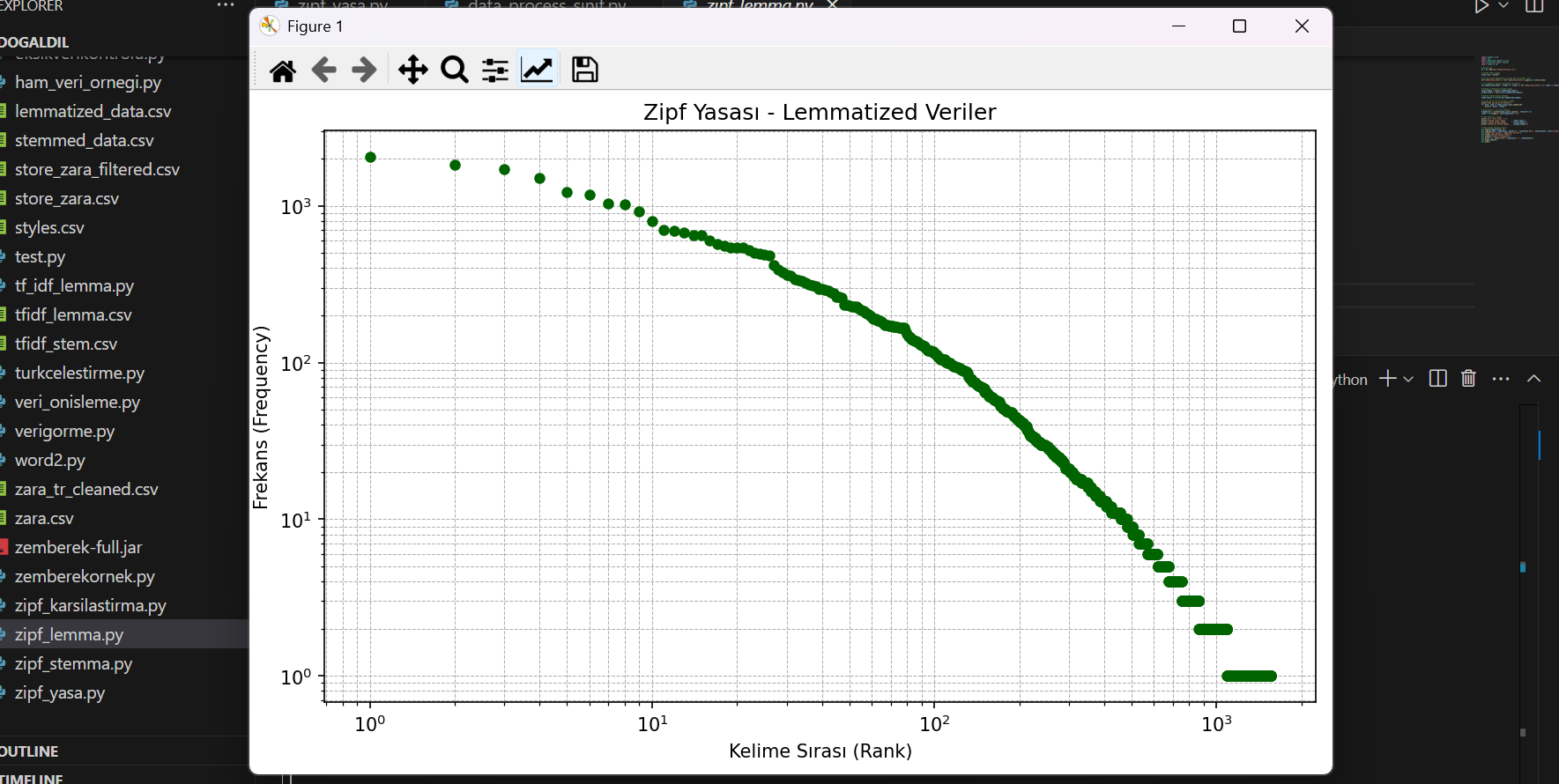
**5. TEMİZLENMİŞ VERİ SETİ ÇIKTISI VE ZİPF ANALİZİ**

Ön işleme adımları sonrasında oluşturulan iki dosyanın linkine aşağıdaki bağlantılar ile ulaşabilirsiniz:

Lemmatization sonucu elde edilen veri: [homework/data/processveri/lemmatized\_data.csv at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/data/processveri/lemmatized_data.csv)

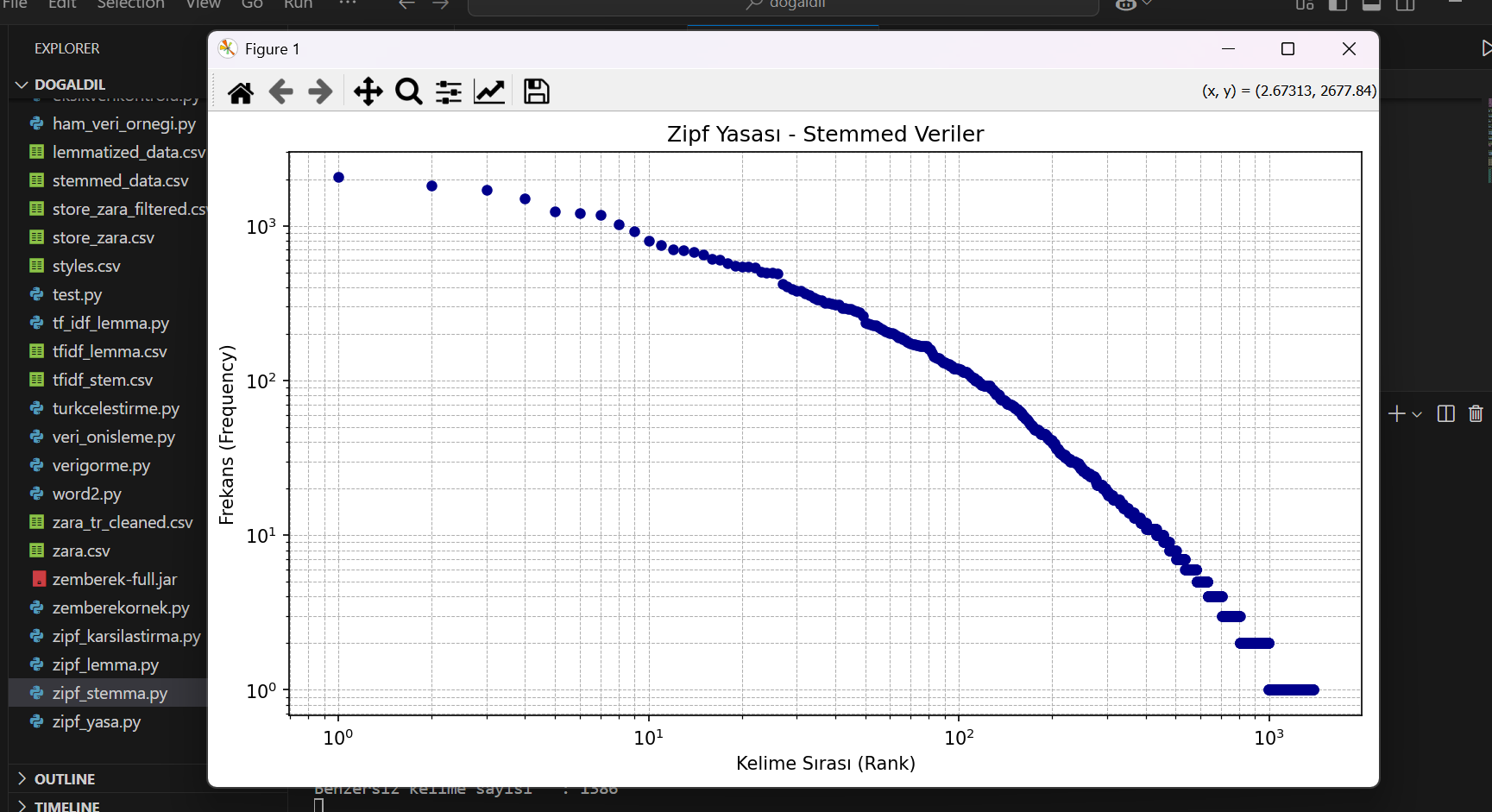
Stemming sonucu elde edilen veri: [homework/data/processveri/stemmed\_data.csv at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/data/processveri/stemmed_data.csv)

**Lemmatized Veriler için Zipf Grafiği:**



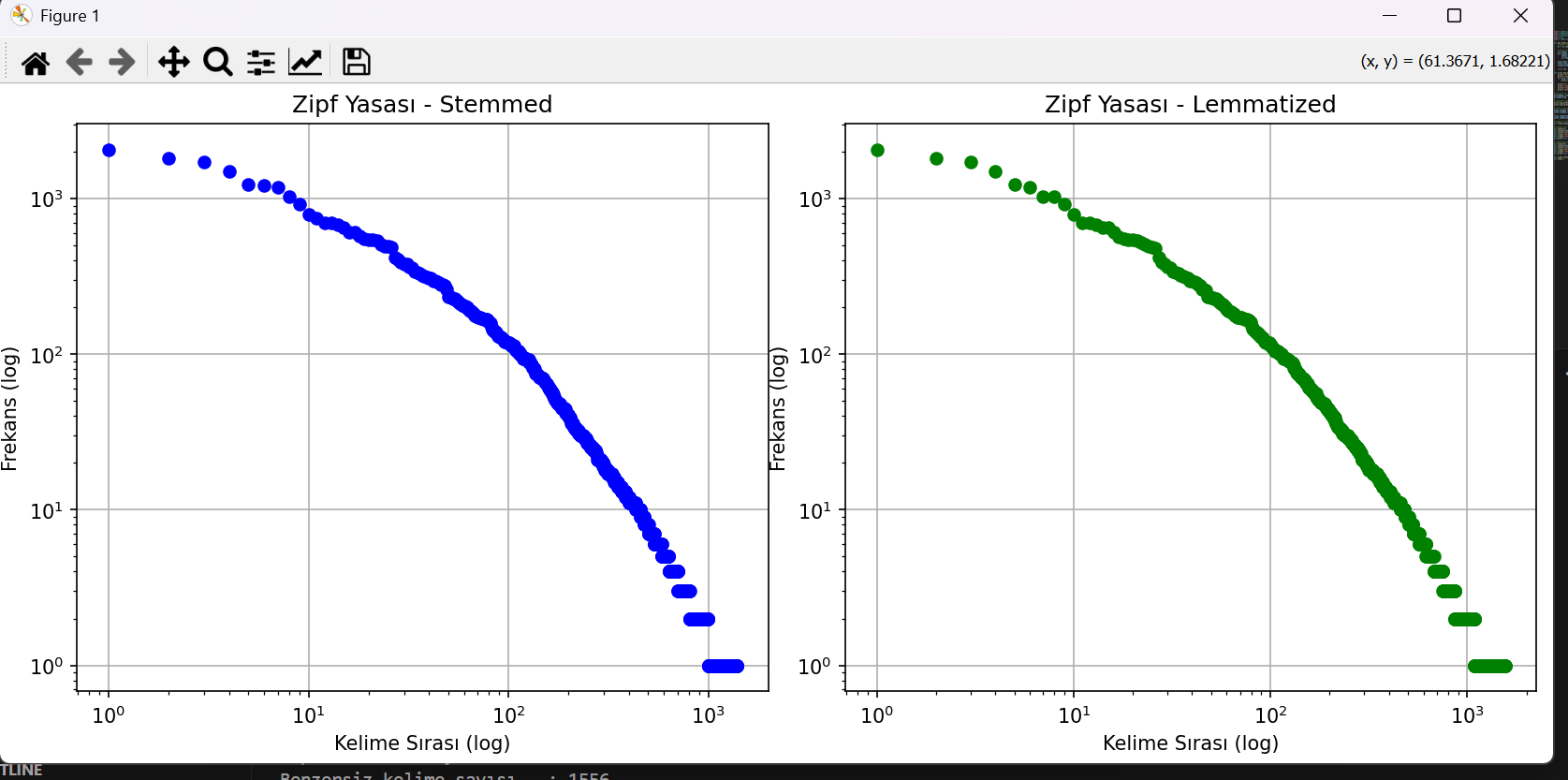
Yukarıda görülen grafik için yazılan kodlara bu link üzerinden erişebilirsiniz: [homework/src/zipf\_lemma.py at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/src/zipf_lemma.py)

**Stemmed Veriler için Zipf Grafiği:**



Yukarıda grafik için yazılan kodlara bu link üzerinden erişebilirsiniz: [homework/src/zipf\_stemma.py at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/src/zipf_stemma.py)

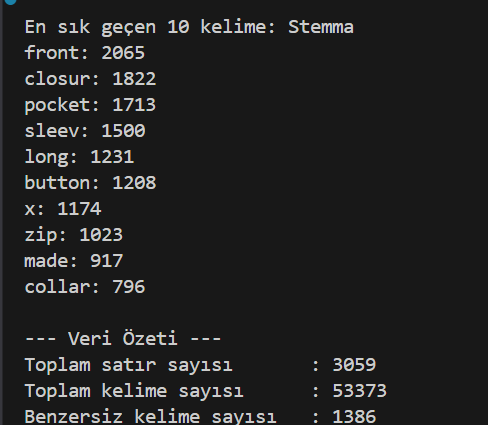
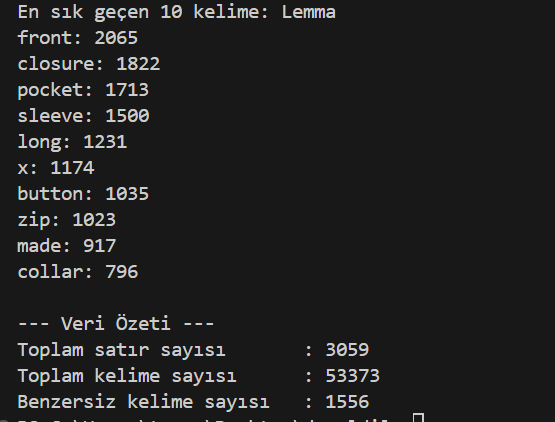
**Her iki grafiğin karşılaştırılması:**



Şekil 16. Stemma ve Lemma Grafik Karşılaştırma

Yukarıda grafik için yazılan kodlara bu link üzerinden erişebilirsiniz: [homework/src/zipf\_karsilastirma.py at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/src/zipf_karsilastirma.py)

Her iki grafikte de Zipf yasasına uygun şekilde log-log eksenlerde yaklaşık olarak doğrusal bir ilişki görülmektedir.

**En Sık Geçen Kelimeler Karşılaştırması**

Şekil 17. Lemma En Sık Geçen Kelimeler Şekil 18. Stemma En Sık Geçen Kelimeler

Şekil 17 ve Şekil 18 de görüldüğü üzere Lemmatization ve Stemming sonrası en sık geçen kelimeler büyük oranda benzerdir. Ancak stemming işlemi bazı kelimeleri dilbilgisel olarak bozarak (örneğin closure → closur, sleeve → sleev) daha az benzersiz kelime üretmiştir.

Her iki yöntemde toplam kelime sayısı aynı kalmış; ancak stemming yöntemi, benzersiz kelime sayısını daha da azaltarak daha yoğun bir temsil sağlamıştır. Zipf dağılımı her iki yöntemde de geçerliliğini korumuştur.

Kısaca stemming işlemi, bazı kelimeleri bozar nitelikte (örnek: "closure" → "closur") çıktılar vermiştir. Bu yüzden çok az da olsa yapısal bozulmalar gözlenmiştir.

**6. VEKTÖRLEŞTİRME İŞLEMLERİ**

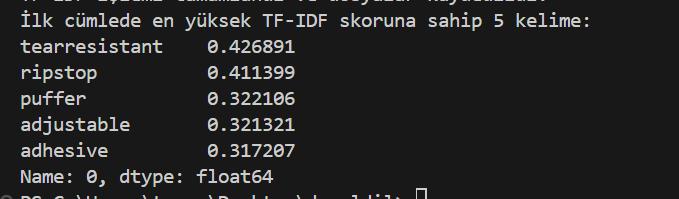
**6.1 TF-IDF VEKTÖRLEŞTİRME**

TF-IDF vektörleştirme işlemi sonucunda oluşan iki ayrı csv dosyasının konumları aşağıdaki linklerde verilmiştir.

tf\_idf\_lemmazited.csv 🡪 [homework/data/processveri/tfidf\_lemmatized.csv at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/data/processveri/tfidf_lemmatized.csv)

tf\_idf\_stemmed.csv🡪 [homework/data/processveri/tfidf\_stemmed.csv at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/data/processveri/tfidf_stemmed.csv)

Ayrıca bu iki csv dosyasını oluşturmak için kullandığım kodlara bu link üzerinden erişebilirsiniz: [homework/src/tf\_idf.py at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/src/tf_idf.py)

****

Şekil 19. TF-IDF Örnek Kod Çıktısı

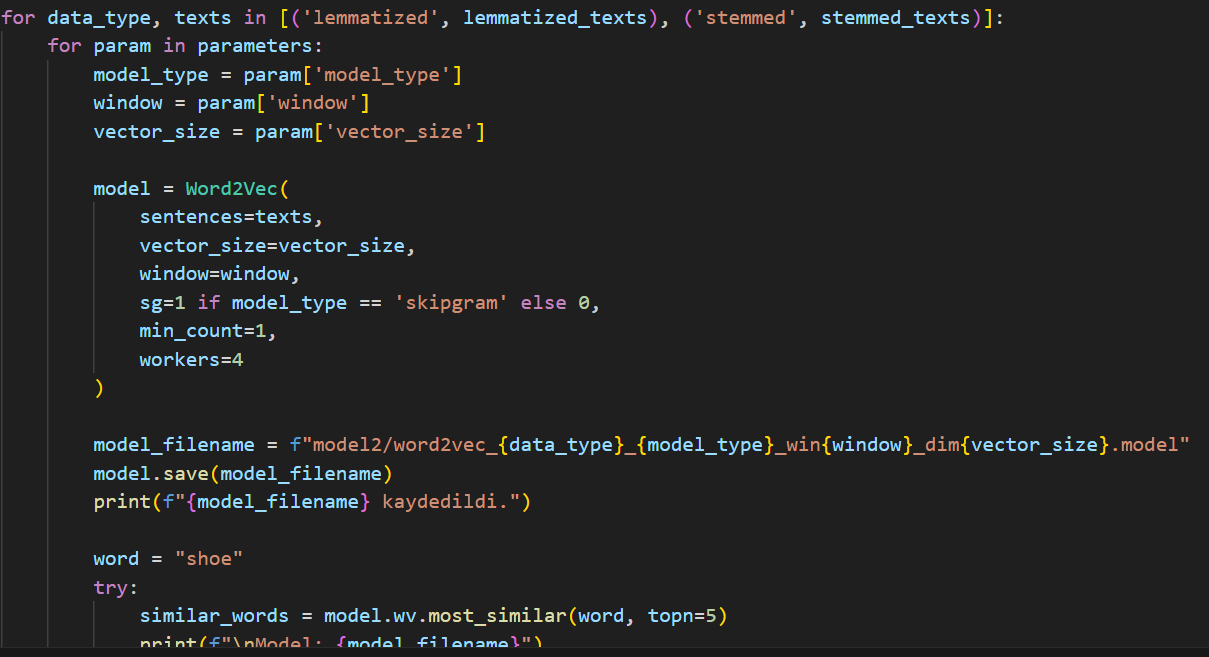
Şekil 19 da görüldüğü üzere TF-IDF vektörleştirme işlemi tamamlandıktan sonra ilk cümlenin kelimelerinin TF-IDF değerleri örnek olarak verilmiştir.

**6.2 WORD2VEC VEKTÖRLEŞTİRME**

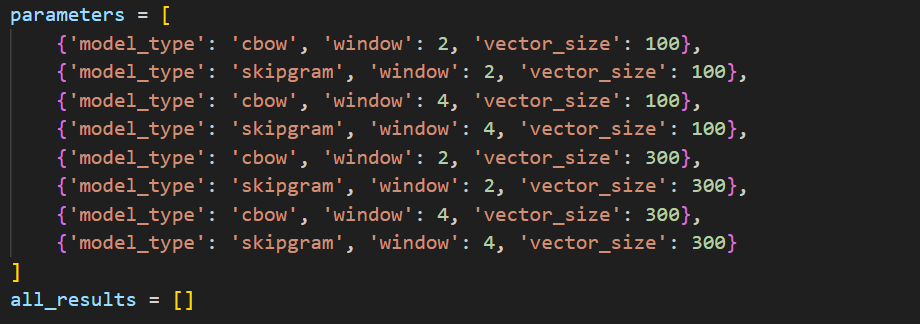
Hem lemmatize edilmiş hem de stemmed versiyonlar üzerinde CBOW ve Skip-gram algoritmaları, farklı pencere ve vektör boyutları ile ayrı ayrı eğitilmiştir.

İlgili Link: [homework/src/word2.py at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/src/word2.py)

Belirli parametrelerle eğitilen her model dosyasını /models klasörü altında bulabilirsiniz: [homework/models at main · tucecnt/homework](https://github.com/tucecnt/homework/tree/main/models)

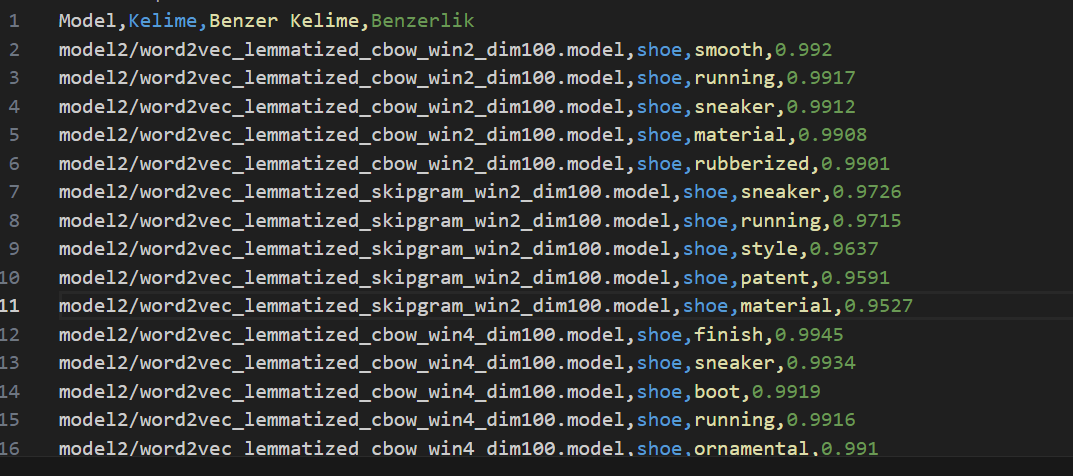


Şekil 20. Model Eğitimi



Şekil 21. Model Parametreleri

Deneme amaçlı her model için belirli bir kelimeye göre en yakın 5 anlamlı kelimenin sonuçlarını ise bu bağlantıya sahip csv dosyasına kaydettim (Hem lemma hem stemma için): [homework/report/similar\_words\_report.csv at main · tucecnt/homework](https://github.com/tucecnt/homework/blob/main/report/similar_words_report.csv)



Şekil 21. Model örnekleri csv dosyası

Model yapılandırmalarını inceleyeceğimiz zaman; CBOW modelleri genellikle daha yüksek benzerlik skorları üretmiş. 300 boyutlu modeller, 100 boyutlu modellere göre daha yüksek benzerlik değerleri gösteriyor.

Csv dosyasını incelediğimizde çıkan sonuçlara göre En yüksek benzerlik skorları genellikle şu modellerden gelmiştir:

* word2vec\_stemmed\_cbow\_win4\_dim300.model → shoe - run: **0.9981**
* word2vec\_lemmatized\_cbow\_win2\_dim300.model → shoe - monochromatic: **0.9979**
* word2vec\_stemmed\_cbow\_win2\_dim300.model → shoe - smooth: **0.9977**

Bu yüksek skorlar, özellikle “CBOW + 300 boyut + win4” parametrelerinin kelime bağlamını iyi öğrendiğini göstermektedir.

**7. SONUÇ**

Bu ödev, metin tabanlı veri üzerinde farklı doğal dil işleme ve vektörleştirme yöntemlerinin uygulanmasıyla, Zipf yasasının geçerliliğini ve model performanslarını incelememe olanak sağlamıştır. Stemming, veri boyutunu daha da küçültse de anlamsal kayıplar yaratmaktadır. Word2Vec + Lemmatization yönteminin hem anlamsal hem sayısal olarak en uygun sonuçları verdiği tespit edilmiştir.