Metin Ön İşleme

İşleyeceğimiz metinler bu veride reviewText Kolonunda yer almaktaktadır.

Hepsini küçük harf yapma

```
df['reviewText'] = df['reviewText'].str.lower()
```

Noktalama ve diğer işaretlerin kaldırılması

```
df['reviewText'] = df['reviewText'].str.replace('[^\w\s]', '')
```

Metindeki sayılar işimize yaramıyor ise

```
df['reviewText'] = df['reviewText'].str.replace('\d', '')
```

Stopwords istediğimiz kelimeleri anlamsız buldgumuz şeyleri emojileri vb. kaldırma

İngilizce için nltk stopwords kelimeleri seçen bir kütüphane var

```
    nltk.download('stopwords')
```

```
df['reviewText'] = df['reviewText'].apply(lambda x: " ".join(x for x in str(x).split() if x not in sw))
```

Rarewords

```
temp_df = pd.Series(' '.join(df['reviewText']).split()).value_counts()

drops = temp_df[temp_df <= 1]

df['reviewText'] = df['reviewText'].apply(lambda x: " ".join(x for x in x.split() if x not in drops))
```

Tokenization

```
# nltk.download("punkt") Hazır bir tokunlaştırma küüpthanesi inglizce için

df["reviewText"].apply(lambda x: TextBlob(x).words).head()
```

Lemmatization (Köklerine ayırma)

```
# nltk.download('wordnet')
df['reviewText'] = df['reviewText'].apply(lambda x: " ".join([Word(word).lemmatize() for word in x.split()]))
```

Metin Görselleştirme

Metinleri numerik formata getirmemiz lazım.

Terim Frekanslarının Hesaplanması

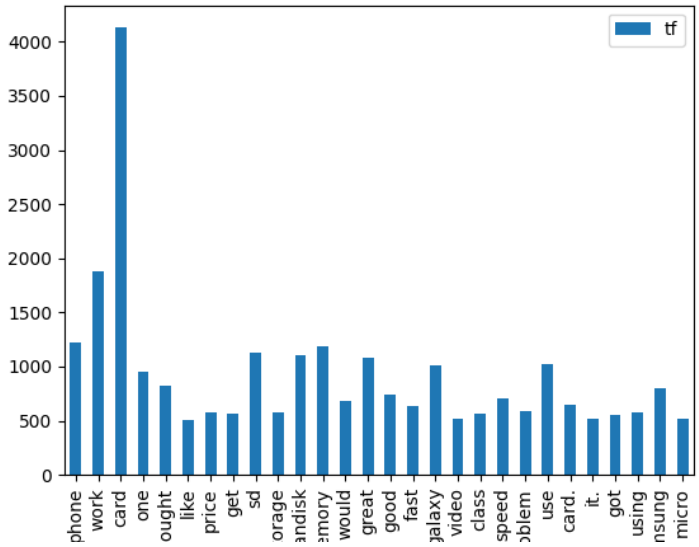
```
tf = df["reviewText"].apply(lambda x: pd.value_counts(x.split(" 
"))).sum(axis=0).reset_index()

tf.columns = ["words", "tf"]

tf.sort_values("tf", ascending=False)
```

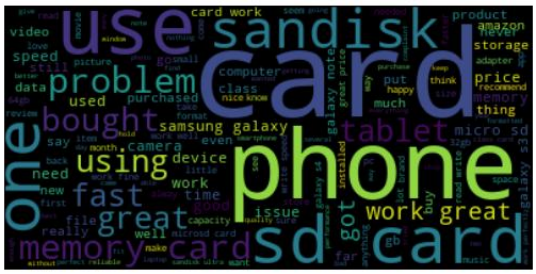
Barplot

```
tf[tf["tf"] > 500].plot.bar(x="words", y="tf") #frekansı 500 den büyükleri getir
plt.show()
```



Wordcloud

```
wordcloud = WordCloud().generate(text)
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```




```
wordcloud = WordCloud(max_font_size=50,
                      max_words=100,
                      background_color="white").generate(text)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Şablonlara Göre Wordcloud

```
tr_mask = np.array(Image.open("tr.png"))

wc = WordCloud(background_color="white",
               max_words=1000,
               mask=tr_mask,
               contour_width=3,
               contour_color="firebrick")

wc.generate(text)
```

```
plt.figure(figsize=[10, 10])
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Duygu Analizi

```
# nltk.download('vader_lexicon')

sia = SentimentIntensityAnalyzer()
sia.polarity_scores("The film was awesome")
{'neg': 0.0, 'neu': 0.423, 'pos': 0.577, 'compound': 0.6249}

sia.polarity_scores("I liked this music but it is not good as the other one")

{'neg': 0.207, 'neu': 0.666, 'pos': 0.127, 'compound': -0.298}
```

Kelimeleri Vektorize Etmenin Yolları

Adım 1: Count Vectorizer'ı Hesapla
(Kelimelerin her bir dokümandaki frekansı)

	this	is	the	first	document	second	and	third	one
This is the first document	1	1	1	1	1				
This document is the second document	1	1	1		2	1			
And this is the third one	1	1	1				1	1	1
Is this the first document	1	1	1	1	1				

Count Vectors: from sklearn.feature_extraction.text import CountVectorizer

```
corpus = ['This is the first document.',
      'This document is the second document.',
      'And this is the third one.',
      'Is this the first document?']

# word frekans
vectorizer = CountVectorizer()
X_c = vectorizer.fit_transform(corpus)
vectorizer.get_feature_names_out()
X_c.toarray()

        array(['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third',

            'this'], dtype=object)

        X_c.toarray()

        array([[0, 1, 1, 1, 0, 0, 1, 0, 1], and yok

            [0, 2, 0, 1, 0, 1, 1, 0, 1], document kelimesi 2 kere geçiyor

            [1, 0, 0, 1, 1, 0, 1, 1, 1], and kelimesi var

            [0, 1, 1, 1, 0, 0, 1, 0, 1]]) and yok
```

 # n-gram frekans

```
vectorizer2 = CountVectorizer(analyzer='word', ngram_range=(2, 2))
X_n = vectorizer2.fit_transform(corpus)
vectorizer2.get_feature_names_out()
X_n.toarray()
```

```
vectorizer2.get_feature_names_out()

array(['and this', 'document is', 'first document', 'is the', 'is this',
    'second document', 'the first', 'the second', 'the third',
    'third one', 'this document', 'this is', 'this the'], dtype=object)

X_n.toarray()

array([[0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0],
    [0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0],
    [1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0],
    [0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1]])
```

 # TF-IDF

Adım 2: TF - Term Frequency'yi Hesapla
(t teriminin ilgili dokümandaki frekansı / dokümandaki toplam terim sayısı)

	this	is	the	first	document	second	and	third	one
This is the first document	0,2	0,2	0,2	0,2	0,2	0	0	0	0
This document is the second document	0,167	0,1667	0,1667	0	0,333333	0,1667	0	0	0
And this is the third one	0,167	0,1667	0,1667	0	0	0	0,17	0,17	0,17
Is this the first document	0,2	0,2	0,2	0,2	0,2	0	0	0	0

Adım 3: IDF - Inverse Document Frequency'i Hesapla
1+ loge ((toplam döküman sayısı+1) / (içinde t terimi olan döküman sayısı+1))

Toplam doküman sayısı: 4

	this	is	the	first	document	second	and	third	one
İçinde t terimi olan döküman sayısı	4	4	4	2	3	1	1	1	1

	this	is	the	first	document	second	and	third	one
IDF	1,0000	1,0000	1,0000	1,5108	1,2231	1,9163	1,9163	1,9163	1,9163

Adım 4: TF * IDF'i Hesapla

	this	is	the	first	document	second	and	third	one
This is the first document	0,2	0,2	0,2	0,3	0,244629	0	0	0	0
This document is the second document	0,167	0,1667	0,1667	0	0,407715	0,3194	0	0	0
And this is the third one	0,167	0,1667	0,1667	0	0	0	0,32	0,32	0,32
Is this the first document	0,2	0,2	0,2	0,3	0,244629	0	0	0	0

Adım 5: L2 Normalizasyonu Yap
Satırların kareleri toplamının karekökünü bul, ilgili satırdaki tüm hücreleri bulduğun değere böl

	this	is	the	first	document	second	and	third	one
This is the first document	0,384	0,3841	0,3841	0,58	0,469791	0	0	0	0
This document is the second document	0,281	0,2811	0,2811	0	0,687624	0,5386	0	0	0
And this is the third one	0,267	0,2671	0,2671	0	0	0	0,51	0,51	0,51
Is this the first document	0,384	0,3841	0,3841	0,58	0,469791	0	0	0	0

```
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_word_vectorizer = TfidfVectorizer()
X_tf_idf_word = tf_idf_word_vectorizer.fit_transform(X)


tf_idf_ngram_vectorizer = TfidfVectorizer(ngram_range=(2, 3))
X_tf_idf_ngram = tf_idf_ngram_vectorizer.fit_transform(X)
```

Sentiment Modeling

Random Forests

```
# Count Vectors
rf_model = RandomForestClassifier().fit(X_count, y)
cross_val_score(rf_model, X_count, y, cv=5, n_jobs=-1).mean()

# TF-IDF Word-Level
rf_model = RandomForestClassifier().fit(X_tf_idf_word, y)
cross_val_score(rf_model, X_tf_idf_word, y, cv=5, n_jobs=-1).mean()

# TF-IDF N-GRAM
rf_model = RandomForestClassifier().fit(X_tf_idf_ngram, y)
cross_val_score(rf_model, X_tf_idf_ngram, y, cv=5, n_jobs=-1).mean()


rf_model = RandomForestClassifier().fit(X_count, y)
cross_val_score(rf_model, X_count, y, cv=5, n_jobs=-1).mean()
np.float64(0.8535096642929808)
rf_model = RandomForestClassifier().fit(X_tf_idf_word, y)
cross_val_score(rf_model, X_tf_idf_word, y, cv=5, n_jobs=-1).mean()
np.float64(0.8341810783316378)
rf_model = RandomForestClassifier().fit(X_tf_idf_ngram, y)
cross_val_score(rf_model, X_tf_idf_ngram, y, cv=5, n_jobs=-1).mean()
np.float64(0.7845371312309257)
```