30. 形態素解析結果の読み込み

解答：

```
with open("./neko.txt.mecab","r") as f:
  text_dict=[]
  sentence_dict=[]
  for line in f.readlines():
    if line=="\n":
     continue
    elif line !="EOS\n":
     node=line.split("\t")
     feature=node[1].split(",")

     if node[0]=="":
      continue

     word_dict={
         "surface":node[0],
         "base":feature[6],
         "pos":feature[0],
         "pos1":feature[1]
     }
     sentence_dict.append(word_dict)
    elif len(sentence_dict)!=0:
      text_dict.append(sentence_dict)
      sentence_dict=[]
print(len(text_dict))
for i in text_dict[:5]:
  for j in i:
    print(j)
  print(" ")
```

実行結果：

```
{'surface': '一', 'base': '一', 'pos': '名詞', 'pos1': '数'}

{'surface': '\u3000', 'base': '\u3000', 'pos': '記号', 'pos1': '空白'}
{'surface': '吾輩', 'base': '吾輩', 'pos': '名詞', 'pos1': '代名詞'}
{'surface': 'は', 'base': 'は', 'pos': '助詞', 'pos1': '係助詞'}
{'surface': '猫', 'base': '猫', 'pos': '名詞', 'pos1': '一般'}
{'surface': 'で', 'base': 'だ', 'pos': '助動詞', 'pos1': '*'}
{'surface': 'ある', 'base': 'ある', 'pos': '助動詞', 'pos1': '*'}
{'surface': '。', 'base': '。', 'pos': '記号', 'pos1': '句点'}

{'surface': '名前', 'base': '名前', 'pos': '名詞', 'pos1': '一般'}
{'surface': 'は', 'base': 'は', 'pos': '助詞', 'pos1': '係助詞'}
{'surface': 'まだ', 'base': 'まだ', 'pos': '副詞', 'pos1': '助詞類接続'}
{'surface': '無い', 'base': '無い', 'pos': '形容詞', 'pos1': '自立'}
{'surface': '。', 'base': '。', 'pos': '記号', 'pos1': '句点'}
```

（結果の一部）

まとめ：

各文を単独でリストの単位とする

31. 動詞

解答：

```
pos_dict=[]
for i in text_dict:
  for j in i:
    if j['pos']=="動詞":
     pos_dict.append(j)
for k in pos_dict[:5]:
  print(k['surface'])
```

実行結果：

```
生れ
つか
し
泣い
し
```

（結果の一部）

まとめ：
list を巡回し、辞書要素の pos タグが「動詞」に対応する場合、この要素を記録する

32. 動詞の原形

解答：

```
pos_dict=[]
for i in text_dict:
  for j in i:
    if j['pos']=="動詞":
     pos_dict.append(j)
for k in pos_dict[:5]:
  print(k['base'])
```

実行結果：

```
生れる
つく
する
泣く
する
```

（結果の一部）

まとめ：

問題を解く考え方は前の問題と同じだ。

33. 「A の B」

解答：

```
A_of_B_dict=[]
for i in text_dict:
  for j in range(len(i)-2):
```

```
    if(i[j]['pos']=="名詞" and i[j+1]['surface'] == "の" and i[j+2]['pos'] == "名詞"):
      A_of_B_dict.append(i[j]['surface']+i[j+1]['surface']+i[j+2]['surface'])
for k in A_of_B_dict[:5]:
  print(k)
```

実行結果：

```
彼の掌
掌の上
書生の顔
はずの顔
顔の真中
```

（結果の一部）

まとめ：

```
i[j]['pos']=="名詞" and i[j+1]['surface'] == "の" and i[j+2]['pos'] == "名詞"
```

以上のコードは【A の B】形式に適合するかどうかを判断するために用いられる

```
i[j]['surface']+i[j+1]['surface']+i[j+2]['surface']
```

該当する場合は、結合された文字列を記録します

34. 名詞の連接

解答：

```
noun_conj=[]
for sentence in text_dict:
  for i,word in enumerate(sentence):
    if(word["pos"]=="名詞")&((i==0)|(sentence[i-1]["pos"]!="名詞")):
      nouns=""
      j=0

      while(i+j<len(sentence)-1)&(sentence[i+j]["pos"]=="名詞"):
        nouns+=sentence[i+j]["surface"]
        j+=1
      if j>=2:
        noun_conj.append(nouns)
print(noun_conj)
```

実行結果：

```
['人間中', '一番獰悪', '時妙', '一毛', 'その後猫', '一度', 'ぶうぶうと煙', '邸内', '三毛', '書生以外',
```

（結果の一部）

まとめ：

私はリストの中で最も長いユニットを探してみましたが、得られた結果は少しおかしいです……

```
manyaslip'twixtthecupandthelip
```

35. 単語の出現頻度

解答：

```
import itertools
from collections import Counter
```

```
flat=list(itertools.chain.from_iterable(text_dict))
flat=[f["base"] for f in flat if f["pos"]!="記号"]


words = Counter(flat)
word_freq=words.most_common()
print(word_freq)
```

実行結果 ：

```
[('の', 9194), ('て', 6848), ('は', 6420), ('に', 6243), ('を', 6071), ('だ', 5972), ('と', 5508), ('が', 5337), ('た', 4267),
```

結果の一部)

まとめ：

list(itertools.chain.from_iterable(text_dict))この関数はネストされた list を list に変換します

```
import itertools
split_task_list = {'train':["MNLI", "MRPC", "SST-2", "QQP", "QNLI", "RTE", "SNLI"], 'val':["MNLI", "MRPC", "SST-2", "QQP", "QNLI", "RTE", "SNLI"]}
qqq=list(itertools.chain.from_iterable(split_task_list.values()))
print(qqq)
```

```
['MNLI', 'MRPC', 'SST-2', 'QQP', 'QNLI', 'RTE', 'SNLI', 'MNLI', 'MRPC', 'SST-2', 'QQP', 'QNLI', 'RTE', 'SNLI']
```

counter 関数統計要素の出現回数、counter タイプの変数を返す

```
from collections import Counter

list_1 = ['hh', "hh", "k", "f"]
counter = Counter(list_1)
print(counter)
```

```
Counter({'hh': 2, 'k': 1, 'f': 1})
```

most_common()関数データを大から小に並べ、リストに戻す

```
from collections import Counter

list_1 = ['hh', "hh", "k", "f"]
counter = Counter(list_1)
counter=counter.most_common()
print(counter)
```

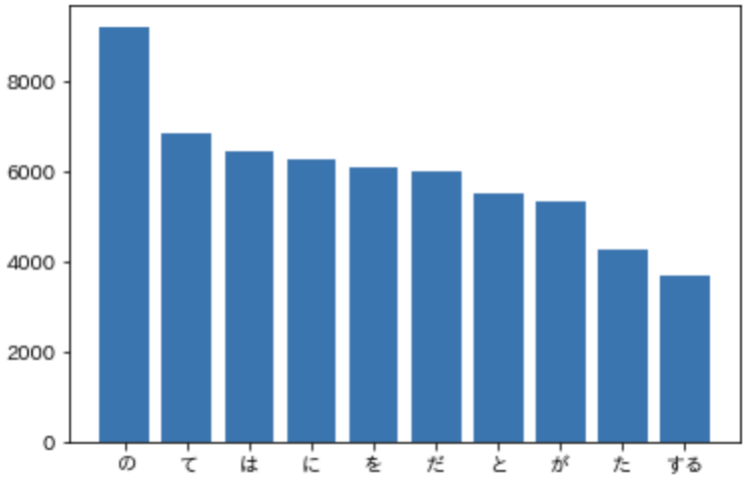
```
[('hh', 2), ('k', 1), ('f', 1)]
```

36.頻度上位 10 語

解答：

```
import matplotlib.pyplot as plt
import japanize_matplotlib

word_freq=dict(word_freq[:10])
plt.figure()
plt.bar(word_freq.keys(),word_freq.values())
plt.show()
```

実行結果 ：



まとめ：

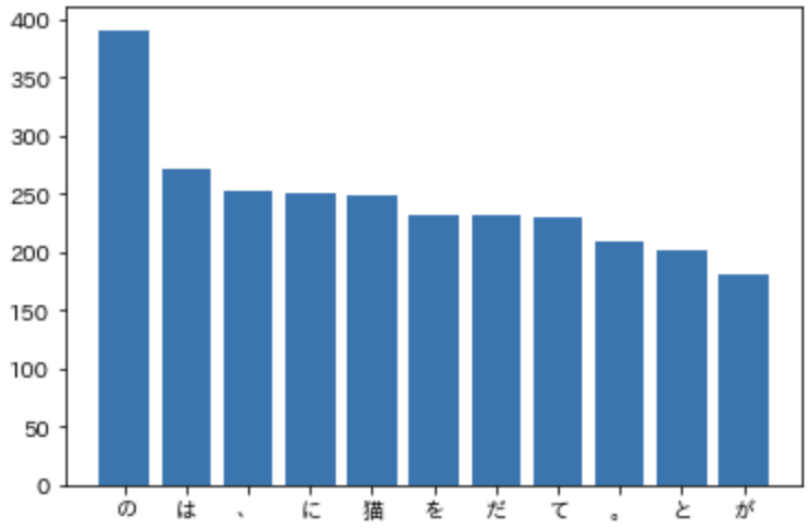
- `plt.bar()` 棒グラムを描く
- `plt.xlabel()` `plt.ylabel()` x 軸、y 軸のマークアップに使用
- `plt.title()` 画像にタイトルを追加するには
- `plt.legend()`凡例とコメントを追加するには
- `plt.show()` 最終プレゼンテーション用画像

37.「猫」と共起頻度の高い上位１０語

解答：

```
from collections import Counter
import matplotlib.pyplot as plt
import japanize_matplotlib

neko_cooc=Counter([])

for sentence in text_dict:
  words=[word["base"]for word in sentence]

  if "猫" in words:
    neko_cooc+=Counter(words)


neko_cooc=dict(neko_cooc.most_common()[:11])
neko_cooc.pop("猫")

plt.figure()
plt.bar(neko_cooc.keys(),neko_cooc.values())
plt.show()
```

実行結果 ：

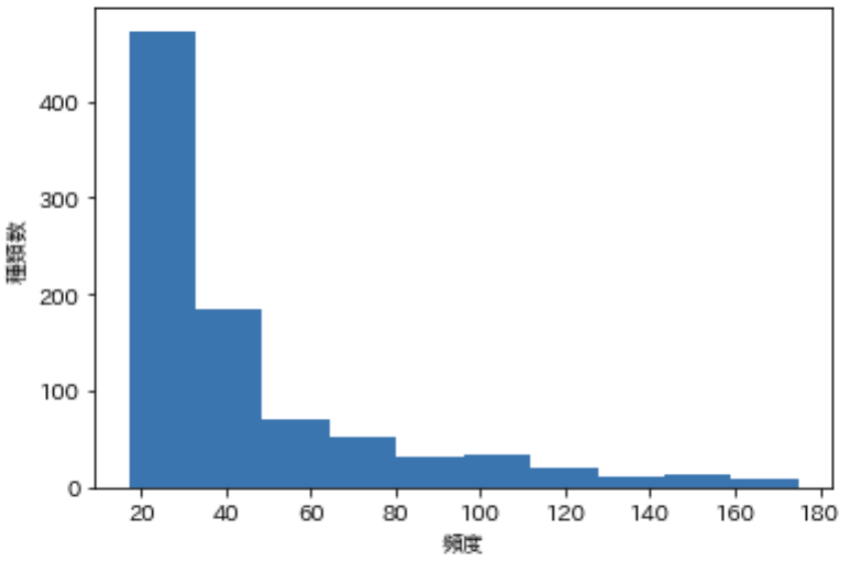


38.ヒストグラム

解答：

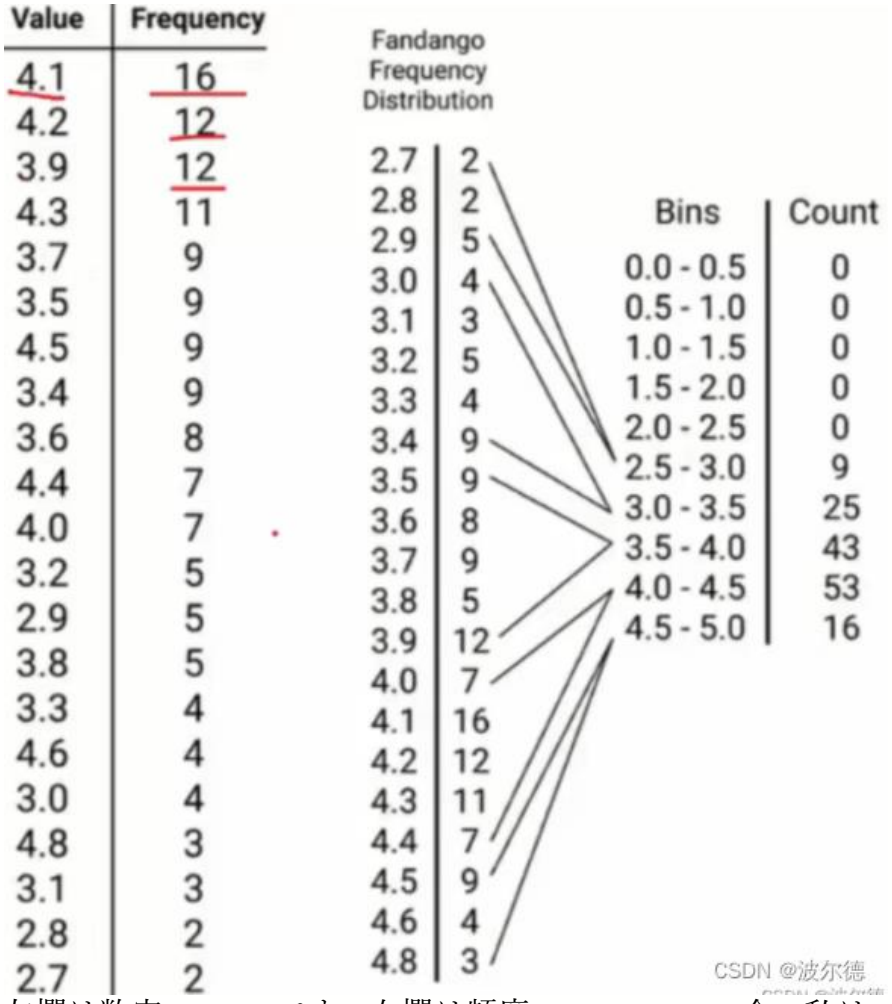
```
word_f=dict(word_freq[100:1000])

plt.figure()
plt.hist(word_f.values())
plt.xlabel("頻度")
plt.ylabel("種類数")
plt.show()
```

実行結果 ：



まとめ：

`plt.hist()`の具体的な役割:



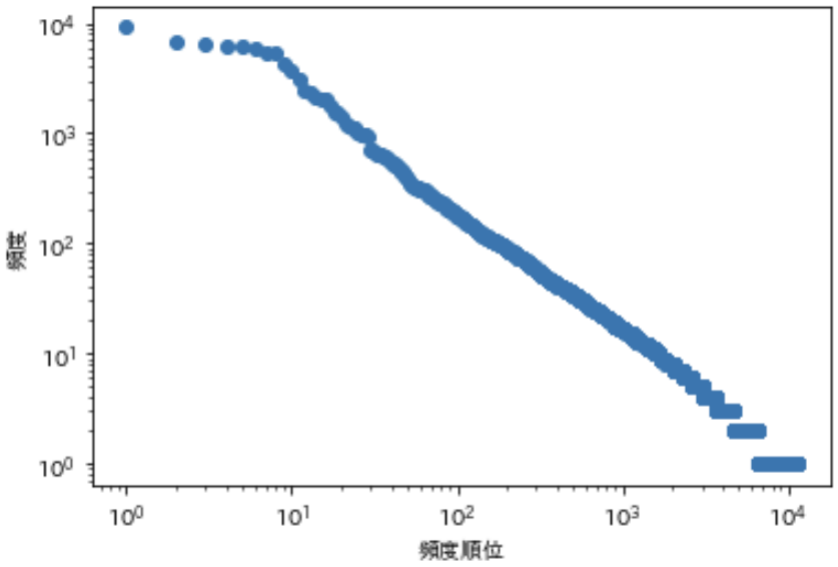
図に示すように、左欄は数字 value です。右欄は頻度 frequency。今、私は 0 ~ 5 の区間を 10 個のビン（箱）に分けて、それぞれの箱の大きさは 0.5 です。図の右端に示すように。
図中の数字が対応する頻度は、Bins の識別された数字の違いに応じて加算されることがわかる。

39.zipf の法則

解答：

```
rank=range(1,len(word_freq)+1)


plt.figure()
plt.scatter(rank,dict(word_freq).values())
plt.xscale("log")
plt.yscale("log")
plt.xlabel("頻度順位")
plt.ylabel("頻度")
plt.show()
```

実行結果 ：



まとめ：

pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, *, data=None, **kwargs)

x，y：サイズが（n，）の配列を表しています。つまり、散点図を描画しようとしているデータ点です

s：実数または配列サイズ（n，）であり、これはオプションのパラメータです。

c:色を表しており、オプションでもあります。デフォルトは青'b'で、マーカーの色を表しています

marker:マーカーのスタイルを表し、デフォルトでは「o」です。

cmap：Colormap エンティティまたは colormap の名前であり、cmap は c が浮動小数点配列である場合にのみ使用されます。説明がなければ image.cmap です

norm：Normalize エンティティは、データ輝度を 0〜1 の間に変換するために使用されます。また、c が浮動小数点数の配列である場合にのみ使用されます。明示されていない場合は、デフォルト colors.Normalize になります。