1-2. 分析設計 - クラスタリング / 1-4. データ分析 - クラスター分析

メディア芸術データベースのデータを利用して，簡単なクラスタリングを行うサンプルコードを見ていこう．

データセットの準備

以下では，メディア芸術データベースのデータを利用し，バンダイビジュアルが提供するアニメビデオパッケージ（DVDソフトやBlu-rayディスクソフトなど、一般・家庭向けに提供されているアニメ作品のビデオソフト）について，価格と収録時間を特徴量としてクラスタリングを行う． 上記のリンクから「アニメビデオパッケージ」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

※「1-2. クラスタリング/1-4. クラスター分析」，「1-2. データの収集、加工、分割／統合」は同じデータセットを利用するため，もし同じものを持っている場合は以下の取得作業は不要である．そちらをアップロードしよう．

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう． （およそ5分程度）

In [ ]:
```
import json
import pandas as pd
!pip install japanize-matplotlib
import japanize_matplotlib
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

json_open = open('metadata_an-item_an202_00001.json', 'r')
json_load = json.load(json_open, strict=False)
```

```
Requirement already satisfied: japanize-matplotlib in /usr/local/lib/python3.10/dist-packages (1.1.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from japanize-matplotlib) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages
```

(from matplotlib->japanize-matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->japanize-matplotlib) (1.16.0)

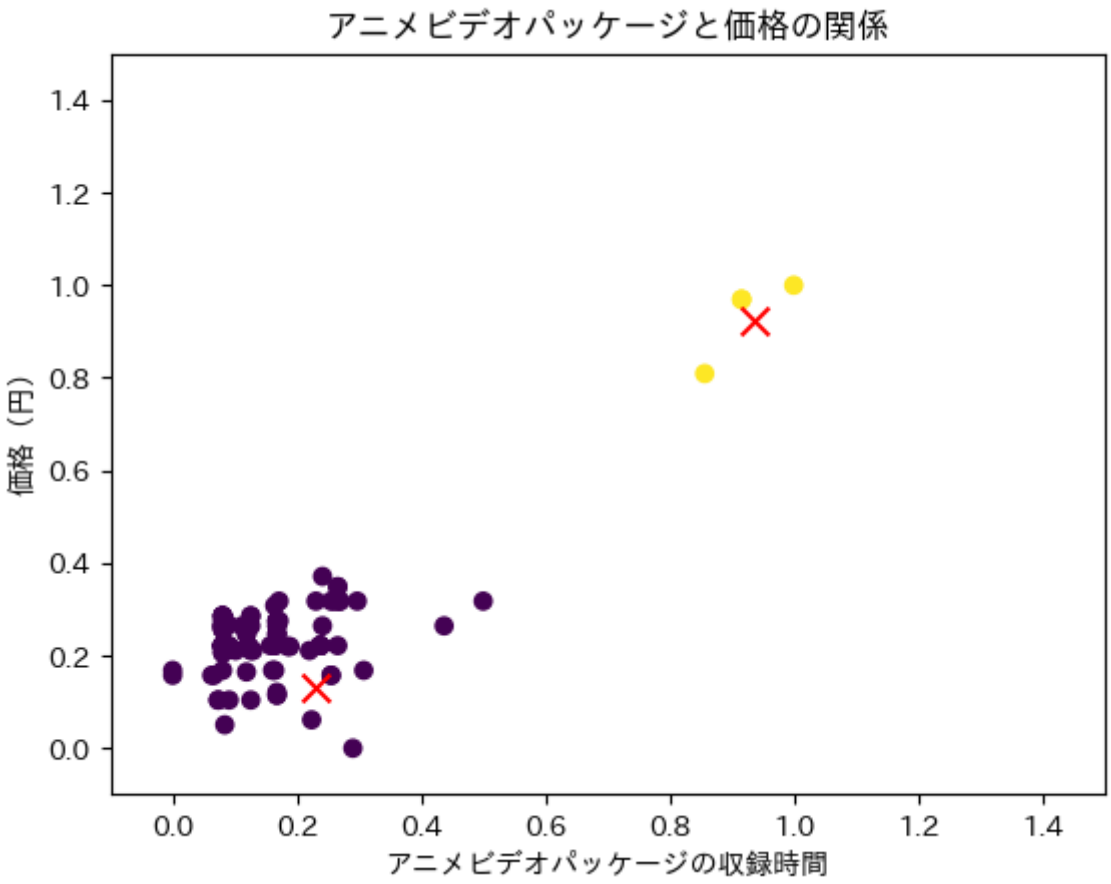
クラスタリング

PythonではScikit-learnというモジュールが提供する `KMeans` を用いることで，簡単にクラスタリングを行うことができる.

In [ ]:
```
#データの前処理
df = pd.DataFrame(json_load["@graph"])
df = df[['materialExtent',"price",'label',"publisher"]]
df = df.dropna(how="any")
df=df[df['publisher'].str.contains("バンダイビジュアル")]

#テキストデータから特徴量を抽出する処理
df['price'] = df['price'].str.extract(r'(\d+)', expand=False) # 一番最初にマッチする数字列のみを抽出
df['long']=df['materialExtent'].str.extract(r'(\d+)分', expand=False)

#欠損値処理
df=df.dropna(how='any')
df['price']=df['price'].astype(int)
df['long']=df['long'].astype(int)
cluster_df = df[["price",'long']]

#データの正規化
for column in cluster_df.columns:
  normalized_data = (cluster_df[column] - cluster_df[column].min()) / (cluster_df[column].max
  cluster_df[column] = normalized_data

# 2つのクラスタに分ける
kmeans = KMeans(n_clusters=2)

#k-meansクラスタリングのフィッティング
kmeans.fit(cluster_df)

#クラスタリング結果の取得
labels = kmeans.labels_  # 各データ点のクラスター割り当て結果
centroids = kmeans.cluster_centers_  # 各クラスターの中心点

#クラスタリング結果の可視化
fig, ax = plt.subplots(dpi=100)
plt.scatter(cluster_df["long"], cluster_df["price"], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='red', s=100)
ax.set_xlim(-0.1, 1.5)
ax.set_ylim(-0.1, 1.5)
ax.set_title('アニメビデオパッケージと価格の関係')
ax.set_xlabel('アニメビデオパッケージの収録時間')
ax.set_ylabel('価格(円)')
plt.show()
```

```
<ipython-input-2-18a65a0b0b8d>:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  cluster_df[column] = normalized_data
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
```

```
o suppress the warning
  warnings.warn(
```



ハイパーパラメータの決定

モデルにおいて，学習の過程で決定するパラメータ（例えばニューラルネットワークモデルの重み係数）のほかに，学習よりも前に決定しておかなければならないパラメータ（例えばkmeans法におけるクラスタ数）が存在する．これをハイパーパラメータと呼ぶ．

上の例ではハイパーパラメータをK=2と先に決定してしまっていたが，メディア芸術データベースのデータを利用して，データを元に適切にハイパーパラメータの決定を行う手法のサンプルコードを見ていこう．

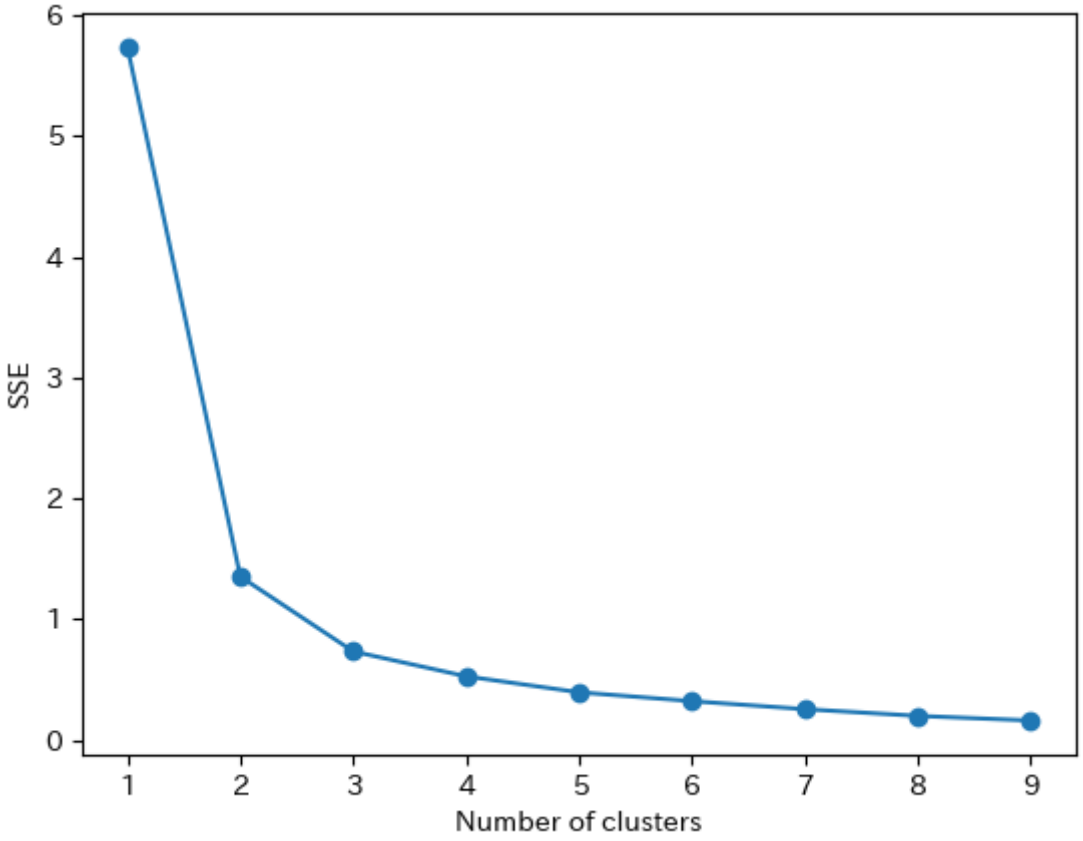
エルボー法

今回はエルボー法という手法を用いてクラスタ数を決定する． エルボー法とは，K個のクラスタに分けたときのSSE（クラスタ内誤差平方和）をプロットし，その値の減り方がガクッと落ちた時（ひじのように曲がっているとき）のクラスタ数を利用するという手法になる． 実際に図を書いてみよう．

In [ ]:
```
distortions = [] # 各クラスタ数におけるSSE(クラスタ内誤差平方和)を保存しておくリスト

for i  in range(1,10):            # 1~9クラスタまでのSSEを一気に計算
    km = KMeans(n_clusters=i)
    km.fit(cluster_df)
    distortions.append(km.inertia_)  # SSEを取得し,リストに追加

# プロット
plt.plot(range(1,10),distortions,marker='o')
plt.xlabel('Number of clusters')
```

```
plt.ylabel('SSE')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
o suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
o suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
o suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
o suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
o suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
o suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
o suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
o suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly t
o suppress the warning
  warnings.warn(
```



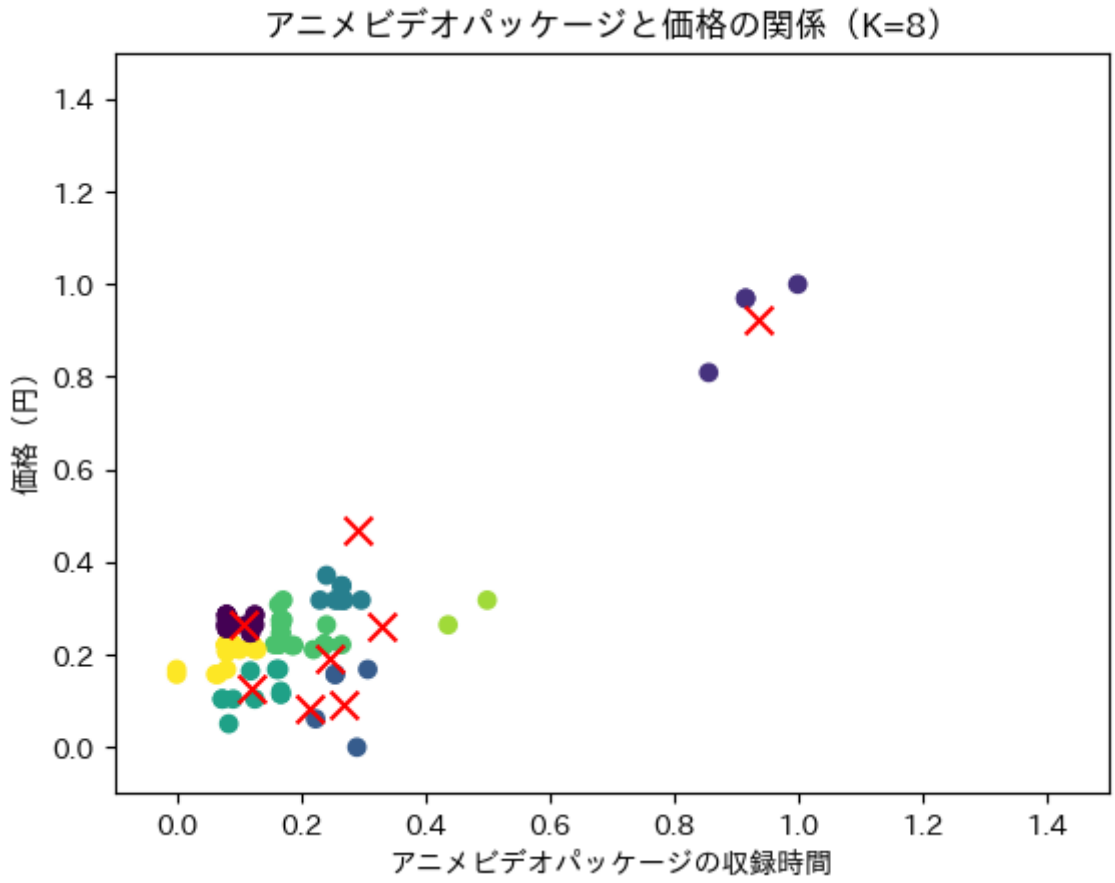
上の図より，クラスタ数は2-3個くらいが適切そうだということがわかる．

プロットして比較

実際に，K＝8としてプロットしたものを，上でプロットしたK=2のものと比較してみよう．

In [ ]:
```
#ハイパーパラメータクラスタ数を8

kmeans = KMeans(n_clusters=8)
#k-meansクラスタリングのフィッティング
kmeans.fit(cluster_df)

#クラスタリング結果の取得
labels = kmeans.labels_  # 各データ点のクラスター割り当て結果
centroids = kmeans.cluster_centers_  # 各クラスターの中心点
#クラスタリング結果の可視化
fig, ax = plt.subplots(dpi=100)
plt.scatter(cluster_df["long"], cluster_df["price"], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='red', s=100)
ax.set_xlim(-0.1, 1.5)
ax.set_ylim(-0.1, 1.5)
ax.set_title('アニメビデオパッケージと価格の関係(K=8)')
ax.set_xlabel('アニメビデオパッケージの収録時間')
ax.set_ylabel('価格(円)')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



見比べてみても，K＝2の方が良さそうな結果になっている．

1-2. 分析設計 - データの収集、加工、分割／統合

分析などのデータは，例えばメディア芸術データベースなどから入手できる．他にも様々なデータベースがあるのでそれを利用することもできる．

以下では，実際に生のjsonデータをダウンロードしてから，どのように目的のデータ（今回は「バンダイビジュアル」が提供するアニメビデオパッケージの，タイトルおよび値段の項目のみに絞ったデータ）になるように加工するのかを見ていく．

データセットの準備

以下では，メディア芸術データベースのデータを利用する． 上記のリンクから「アニメビデオパッケージ」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

※「1-2. クラスタリング/1-4. クラスター分析」，「1-2. データの収集、加工、分割／統合」は同じデータセットを利用するため，もし同じものを持っている場合は以下の取得作業は不要である．そちらをアップロードしよう．

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう．（およそ5分程度）

In [ ]:
```
import json
import pandas as pd

#データ入手
json_open = open('/content/metadata_an-item_an202_00001.json', 'r')
json_load = json.load(json_open, strict=False)
```

加工・分割・統合

In [ ]:
```
#加工
#publisherがバンダイビジュアルのもののみ抽出する
df = pd.DataFrame(json_load["@graph"]).dropna(subset=['publisher', 'label', 'price'])
df=df[df['publisher'].str.contains("バンダイビジュアル")]

#分割
#データをlabelとpriceごとに分割
label_df = df['label']
price_df = df["price"]
print(label_df)
print(price_df)
```

```
3391                                    ヒピラくん
3418                                宇宙戦艦ヤマト2199
3420                            おねがい*ティーチャーツインズ
3444                                Kurobas cup 2013
3445          ラブライブ!school idol project-国立音ノ木坂学院案内-
                  ...
9567                                攻殻機動隊arise
```

```
9577                                攻殻機動隊arise
9703          機動戦士ガンダムUC episode 6「宇宙*と地球*と」*[そら]*[ほし]
9875    (新)テニスの王子様OVA vs genius10 The prince of tennis...
9876                      (新)テニスの王子様 OVA vs genius10
Name: label, Length: 147, dtype: object
3391    3990円 (税込)
3418    3990円 (税込)
3420    7140円 (税込)
3444    8190円 (税込)
3445    6300円 (税込)
        ...
9567       6800円
9577       6800円
9703    6264円 (税込)
9875       6000円
9876       6000円
Name: price, Length: 147, dtype: object
```

In [ ]:

```
#統合
#上のデータを統合
merged_df = pd.concat([label_df, price_df], axis=1)
print(merged_df)
```

```
                                label       price
3391                              ヒピラくん  3990円 (税込)
3418                          宇宙戦艦ヤマト2199  3990円 (税込)
3420                      おねがい*ティーチャーツインズ  7140円 (税込)
3444                     Kurobas cup 2013  8190円 (税込)
3445          ラブライブ!school idol project-国立音ノ木坂学院案内-  6300円 (税込)
...                                ...         ...
9567                          攻殻機動隊arise       6800円
9577                          攻殻機動隊arise       6800円
9703       機動戦士ガンダムUC episode 6「宇宙*と地球*と」*[そら]*[ほし]  6264円 (税込)
9875  (新)テニスの王子様OVA vs genius10 The prince of tennis...       6000円
9876                (新)テニスの王子様 OVA vs genius10       6000円

[147 rows x 2 columns]
```

1-4. データ分析 - ロジスティック回帰分析

ロジスティック回帰モデルとは，二値分類によく使われるモデルであり，ある1つの説明変数xがある目的変数$y=\{0,1\}$のいずれかに対応するとき，$y=0$になる確率を$\frac{exp(a+bx)}{1+exp(a+bx)}$, $y=1$になる確率を$\frac{1}{1+exp(a+bx)}$,というように表すことで予測を行うモデルのことを指す.

ロジスティック回帰分析では，尤度関数の最大化によってモデルのパラメータa,bを決定する.

メディア芸術データベースのデータを利用して，簡単なロジスティック回帰分析のサンプルコードを見ていこう.

データセットの準備

以下では，メディア芸術データベースのデータを利用し，マンガタイトルから「集英社」，「小学館」のどちらが出版したマンガなのかを予測するロジスティック回帰を行う． 上記のリンクから「マンガ単行本」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう.

※「1-4. 単回帰分析」，「1-4. ロジスティック回帰分析」，「1-5. 1〜3次元の図表化」,「1-5. 関係性の可視化（ネットワーク構造）」，「1-7. ソートアルゴリズム」，「1-7. 探索アルゴリズム」，「2-5. データ加工」，「3-3. 機械学習」は同じデータセットを利用するため，もし同じものを持っている場合は以下の取得作業は不要である．そちらをアップロードしよう.

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう． （およそ30分程度）

In [ ]:
```
import json
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

json_open = open('/content/metadata_cm-item_cm101_00001.json', 'r')
json_load = json.load(json_open, strict=False)
```

ロジスティック回帰分析

In [ ]:
```
df_logi = pd.DataFrame(json_load["@graph"])
df_logi=df_logi[['label',"publisher"]]
df_logi["publisher"]=df_logi["publisher"].str[:3]
df_logi=df_logi[(df_logi["publisher"].str.contains("集英社")) |(df_logi["publisher"].str.contains('

# 出版社が「集英社」なら1、それ以外（つまり小学館）なら0に変換する関数
def convert_publisher(publisher):
    if '集英社' in publisher:
        return 1
    else:
        return 0
```

```
# 出版社列を集英社なら1それ以外を0に変換
df_logi['publisher'] = df_logi['publisher'].apply(convert_publisher)

# テキストデータをベクトル化するオブジェクトを作成
vectorizer = TfidfVectorizer()
# テキストデータをベクトル化
features = vectorizer.fit_transform(df_logi['label'])

#訓練データとテストデータに分割
X_train, X_test, Y_train, Y_test = train_test_split(features, df_logi['publisher'], test_size = 0.2,
logistic=LogisticRegression()
#フィッティング
logistic.fit(X_train,Y_train)
#予測
Y_pred = logistic.predict(X_test)
#正解率表示
print('正解率 = ', accuracy_score(y_true=Y_test, y_pred=Y_pred))
```

```
<ipython-input-2-43d7b85fdb78>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gui
de/indexing.html#returning-a-view-versus-a-copy
  df_logi['publisher'] = df_logi['publisher'].apply(convert_publisher)
正解率 =  0.847540185094983
```

1-4. データ分析 - 主成分分析、次元削減

主成分分析（PCA）とは，データの情報をできるだけ失わないように，分散の値ができるだけ大きくなるような新しい軸を取り直すことで次元削減を行う手法である．本来これを行うためには固有値や固有ベクトル計算する必要があるが，PythonではScikit-learnモジュールを利用することで明示的に自分で計算する必要なく簡単にPCAを行うことができる．

メディア芸術データベースのデータを利用して，簡単なPCAのサンプルコードを見ていこう．

ゲームの発売年、商品名の長さ、最大プレイ人数、パッケージの大きさからなるデータの主成分分析

データセットの準備

以下では，メディア芸術データベースのデータを利用する． 上記のリンクから「ゲームパッケージ」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう． （およそ15分程度）

In [ ]:
```
import json
import pandas as pd
import matplotlib.pyplot as plt
!pip install japanize-matplotlib
import japanize_matplotlib

json_open = open('/content/metadata_gm-item_gm301_00001.json', 'r')
json_load = json.load(json_open, strict=False)
df = pd.DataFrame(json_load["@graph"])
```

```
Collecting japanize-matplotlib
  Downloading japanize-matplotlib-1.1.3.tar.gz (4.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.1/4.1 M
B 18.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from japan
ize-matplotlib) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (fr
om matplotlib->japanize-matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from m
atplotlib->japanize-matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (fr
om matplotlib->japanize-matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (fro
m matplotlib->japanize-matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from
matplotlib->japanize-matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (fro
m matplotlib->japanize-matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from m
atplotlib->japanize-matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (fro
m matplotlib->japanize-matplotlib) (3.1.1)
```

```
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->japanize-matplotlib) (1.16.0)
Building wheels for collected packages: japanize-matplotlib
  Building wheel for japanize-matplotlib (setup.py) ... done
  Created wheel for japanize-matplotlib: filename=japanize_matplotlib-1.1.3-py3-none-any.whl size=4120257 sha256=fb31a85665c9f7cbcc8453efddeb8aad9830bb9d983c6a4eccadc103bc62a78b
  Stored in directory: /root/.cache/pip/wheels/61/7a/6b/df1f79be9c59862525070e157e62b08eab8ece27c1b68fbb94
Successfully built japanize-matplotlib
Installing collected packages: japanize-matplotlib
Successfully installed japanize-matplotlib-1.1.3
```

PCA

ゲームの価格，ゲームタイトルの長さ，最大プレイ人数，パッケージサイズの4次元からなるデータについて，主成分分析を行って2次元に圧縮していく．

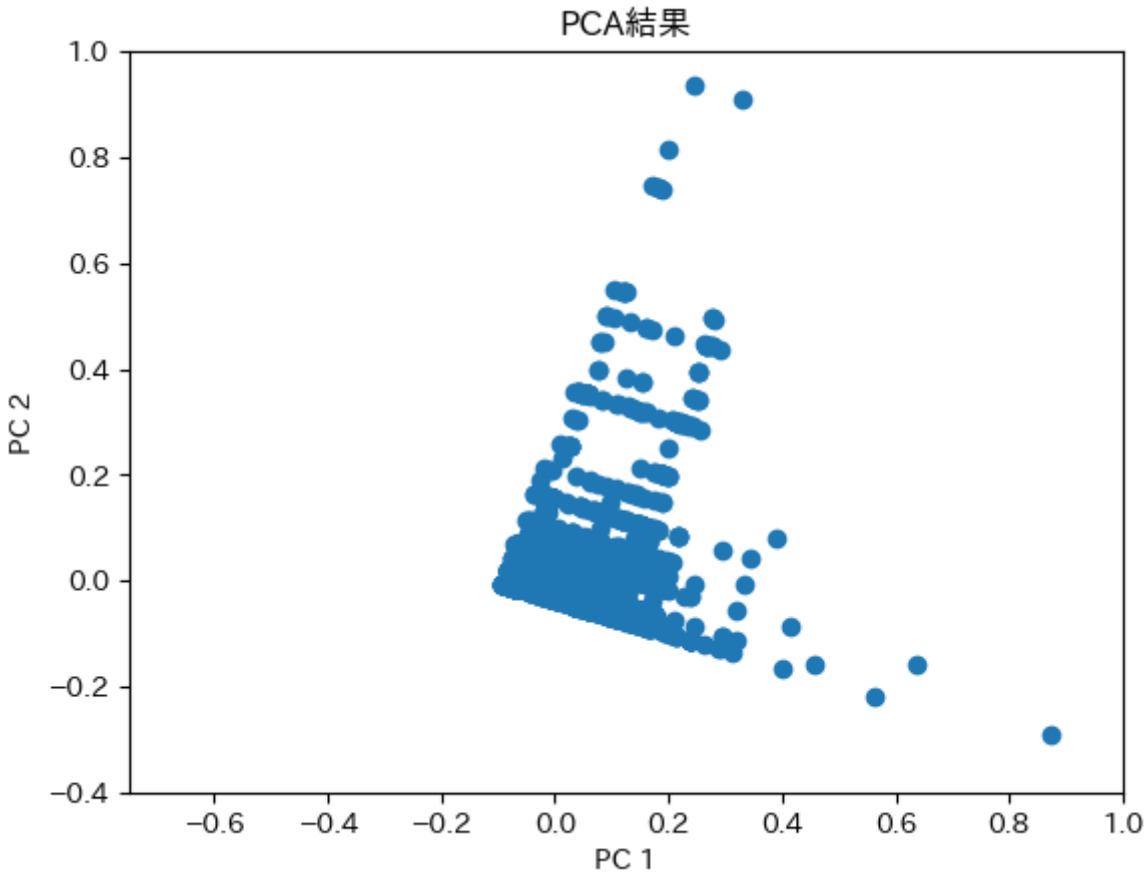
In [ ]:

```python
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.compose import ColumnTransformer
import re
#データの前処理
after_df = pd.DataFrame()
after_df["lenname"]=df['name'].str.len()

def extract_max_number(string):
    numbers = re.findall(r'\d+', string)
    if numbers:
        return max(map(int, numbers))
    else:
        return 0
after_df["player"] =  df['numberOfPlayers'].astype(str).apply(extract_max_number)
after_df["size"] =  df['size'].astype(str).apply(extract_max_number)
after_df["price"] = df['price'].astype(str).apply(extract_max_number)

# 欠損値の処理
after_df.dropna(inplace=True)
for column in after_df.columns:
    after_df[column] = after_df[column].astype(int)
    after_df = after_df[after_df[column] != 0]
after_df = after_df[after_df["player"] < 50]
after_df = after_df[after_df["price"]<100000]

#データの正規化
for column in after_df.columns:
  normalized_data = (after_df[column] - after_df[column].min()) / (after_df[column].max() - af
  after_df[column] = normalized_data

# 主成分分析の実行
pca = PCA()
pca.fit(after_df)

# 分析結果の取得
explained_variance_ratio = pca.explained_variance_ratio_
components = pca.components_

print("寄与率")
print(explained_variance_ratio)
```

```
print("主成分軸")
for i, component in enumerate(components):
    print(f"PC{i+1}: {component}")
```

寄与率
[0.49544559 0.36130937 0.13292933 0.01031571]
主成分軸
PC1: [0.00279892 0.3076039 0.09776951 0.94647405]
PC2: [-0.00298025 0.95147145 -0.02237078 -0.30690837]
PC3: [2.81067594e-04 -8.83243641e-03 9.94957430e-01 -9.99080673e-02]
PC4: [9.99991603e-01 1.97716838e-03 -6.19974722e-04 -3.53571484e-03]

In [ ]:
```
# プロット
fig, ax = plt.subplots(dpi=100)
feature=pca.transform(after_df)
plt.scatter(feature[:, 0], feature[:, 1])
ax.set_xlim(-0.75, 1)
ax.set_ylim(-0.4, 1)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.title('PCA結果')
plt.show()
```



1-4. データ分析 - 単回帰分析

単回帰分析とは，ある1つの説明変数xから1つの目的変数yを導出する回帰モデル$y=ax+b$について，最小二乗法を用いて回帰係数a,bを学習することである．

メディア芸術データベース・ラボ上のデータを利用して，簡単な単回帰分析のサンプルコードを見ていこう．

データセットの準備

以下では，メディア芸術データベースのデータを利用し，バンダイビジュアルが提供するアニメビデオパッケージについて，価格を目的変数，収録時間を説明変数として単回帰分析を行う．上記のリンクから「アニメビデオパッケージ」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

※「1-4. 単回帰分析」，「1-4. ロジスティック回帰分析」，「1-5. 1〜3次元の図表化」，「1-5. 関係性の可視化（ネットワーク構造）」，「1-7. ソートアルゴリズム」，「1-7. 探索アルゴリズム」，「2-5. データ加工」，「3-3. 機械学習」は同じデータセットを利用するため，もし同じものを持っている場合は以下の取得作業は不要である．そちらをアップロードしよう．

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう．（およそ5分程度）

In [ ]:
```
import json
import pandas as pd
from numpy.polynomial import Polynomial
import numpy as np
!pip install japanize-matplotlib
import japanize_matplotlib
import matplotlib.pyplot as plt

json_open = open('/content/metadata_an-item_an202_00001.json', 'r')
json_load = json.load(json_open, strict=False)
```

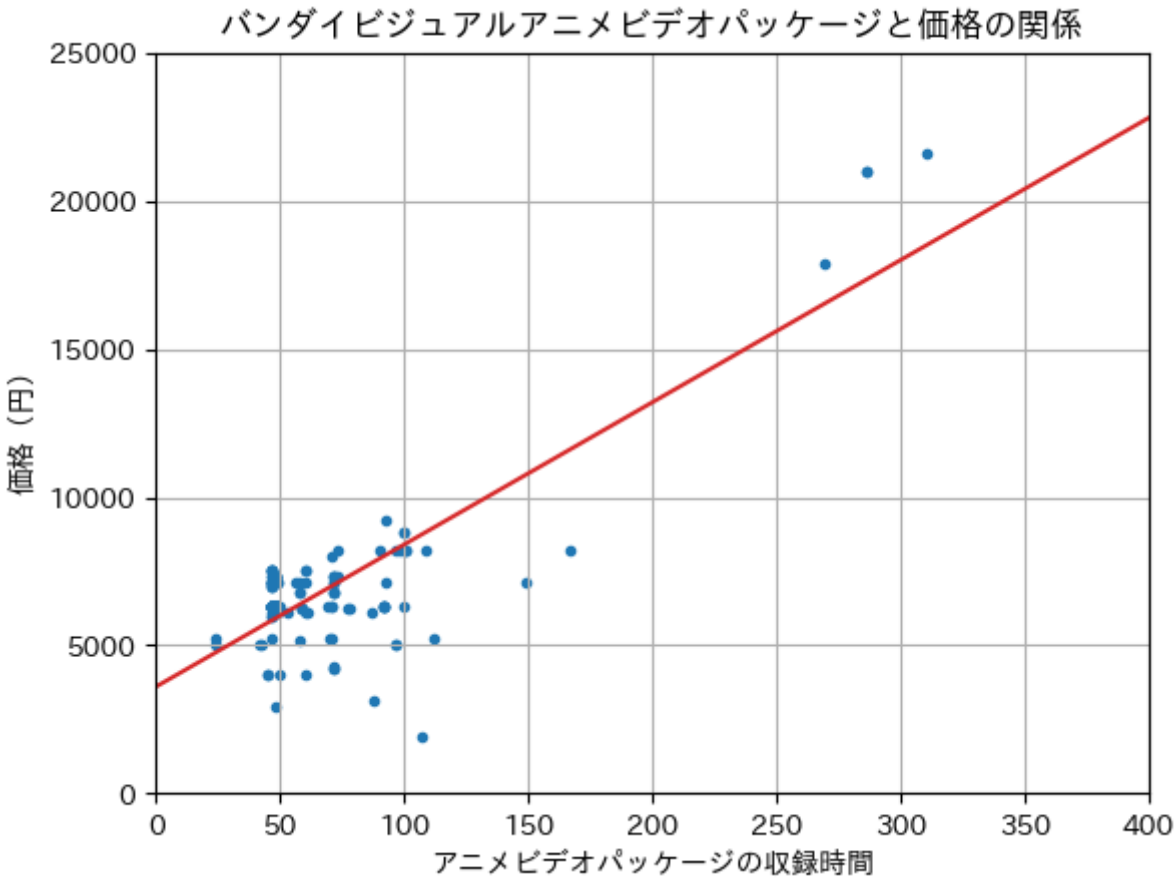
```
Collecting japanize-matplotlib
  Downloading japanize-matplotlib-1.1.3.tar.gz (4.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.1/4.1 M
B 12.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from japan
ize-matplotlib) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (fr
om matplotlib->japanize-matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from m
atplotlib->japanize-matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (fr
om matplotlib->japanize-matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (fro
m matplotlib->japanize-matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from
matplotlib->japanize-matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (fro
```

```
m matplotlib->japanize-matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from m
atplotlib->japanize-matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (fro
m matplotlib->japanize-matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages
(from matplotlib->japanize-matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python
-dateutil>=2.7->matplotlib->japanize-matplotlib) (1.16.0)
Building wheels for collected packages: japanize-matplotlib
  Building wheel for japanize-matplotlib (setup.py) ... done
  Created wheel for japanize-matplotlib: filename=japanize_matplotlib-1.1.3-py3-none-any.whl siz
e=4120257 sha256=fa120325802f86236a562a76ba504dc3d354ffb5c3b9bb3f5c2f11245e
6df6aa
  Stored in directory: /root/.cache/pip/wheels/61/7a/6b/df1f79be9c59862525070e157e62b
08eab8ece27c1b68fbb94
Successfully built japanize-matplotlib
Installing collected packages: japanize-matplotlib
Successfully installed japanize-matplotlib-1.1.3
```

単回帰分析

In [ ]:

```python
#データの前処理
df = pd.DataFrame(json_load['@graph'])
df = df[['materialExtent','price','label','publisher']].dropna(how='any') #欠損値も処理
df=df[df['publisher'].str.contains('バンダイビジュアル')]
#文字列処理
df['price'] = df['price'].str.extract(r'(\d+)', expand=False)
df['long']=df['materialExtent'].str.extract(r'(\d+)分', expand=False)

df['price']=df['price'].astype(int)
df['long']=df['long'].astype(int)

#グラフ表示する関数
def show_graph(X, Y, x, y):
    fig, ax = plt.subplots(dpi=100)
    ax.scatter(X, Y, marker='.')
    ax.plot(x, y, 'tab:red')
    ax.set_title('バンダイビジュアルアニメビデオパッケージと価格の関係')
    ax.set_xlabel('アニメビデオパッケージの収録時間')
    ax.set_ylabel('価格(円)')
    ax.set_xlim(0, 400)
    ax.set_ylim(0, 25000)
    ax.grid()
    plt.show()

#回帰直線をフィッティングする
W = Polynomial.fit(df['long'], df['price'], 1)
result_x = np.linspace(0, 1200, 100)
result_y = W(result_x)

#プロット
show_graph(df['long'], df['price'], result_x, result_y)
```


バンダイビジュアルアニメビデオパッケージと価格の関係
25000
20000
15000
10000
5000
0
価格（円）
0
50
100
150
200
250
300
350
400
アニメビデオパッケージの収録時間

1-5. データ可視化 - 1〜3次元の図表化（棒グラフ、折線グラフ、積み上げ縦棒グラフ）

データと一口に言っても様々なデータが存在し，それをわかりやすく可視化するための手法もデータの中身によって選択する必要がある． メディア芸術データベースのデータを利用して，様々なデータの可視化を行なってみよう．

In [ ]:
```
import json
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
!pip install japanize-matplotlib
import japanize_matplotlib
```

```
Requirement already satisfied: japanize-matplotlib in /usr/local/lib/python3.10/dist-packages (1.1.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from japanize-matplotlib) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->japanize-matplotlib) (1.16.0)
```

棒グラフ

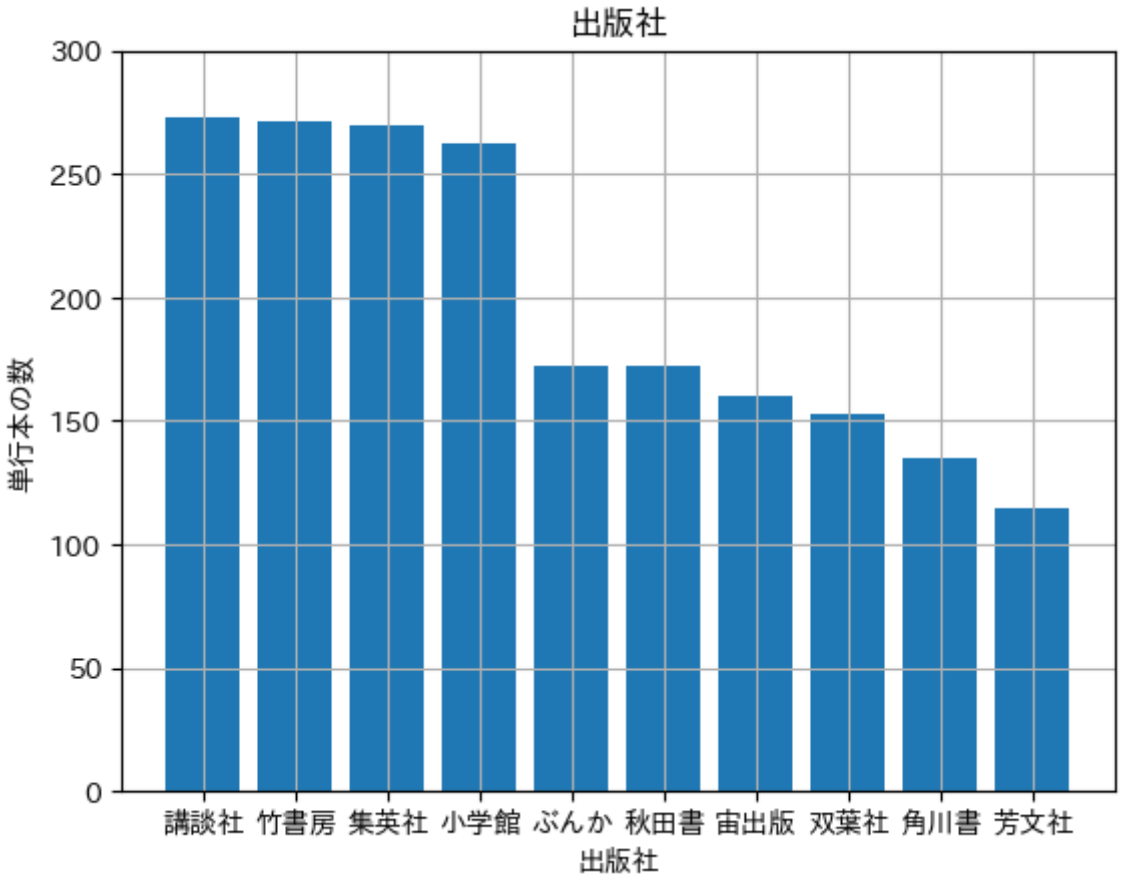
データセットの準備

以下では，メディア芸術データベースのデータを利用し，出版社ごとの出版したマンガ雑誌の種類の累計，およびゲームプラットフォームごとの対応ソフトの種類の累計を可視化する． 上記のリンクから「マンガ雑誌」の「.json」ファイルと，「ゲームバリエーション」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

出版社ごとの出版したマンガ雑誌の種類の累計

In [ ]:
```
json_open = open('/content/metadata_cm-col_cm105_00001.json', 'r')
json_load = json.load(json_open, strict=False)
df = pd.DataFrame(json_load["@graph"])
```

In [ ]:
```
#出版社を数える
count = df['publisher'].str.removeprefix('[発行]').str.removeprefix('[発売]').str[:3]
count=count.value_counts()
#グラフの関数
def show_graph(x, y):
    fig, ax = plt.subplots(dpi=100)
    ax.set_title('出版社')
    ax.set_xlabel('出版社')
    ax.set_ylabel('単行本の数')
    ax.set_ylim(0, 300)
    ax.grid()
    plt.bar(x,y)

#グラフの表示
show_graph(count.index[0:10],count[0:10])
```

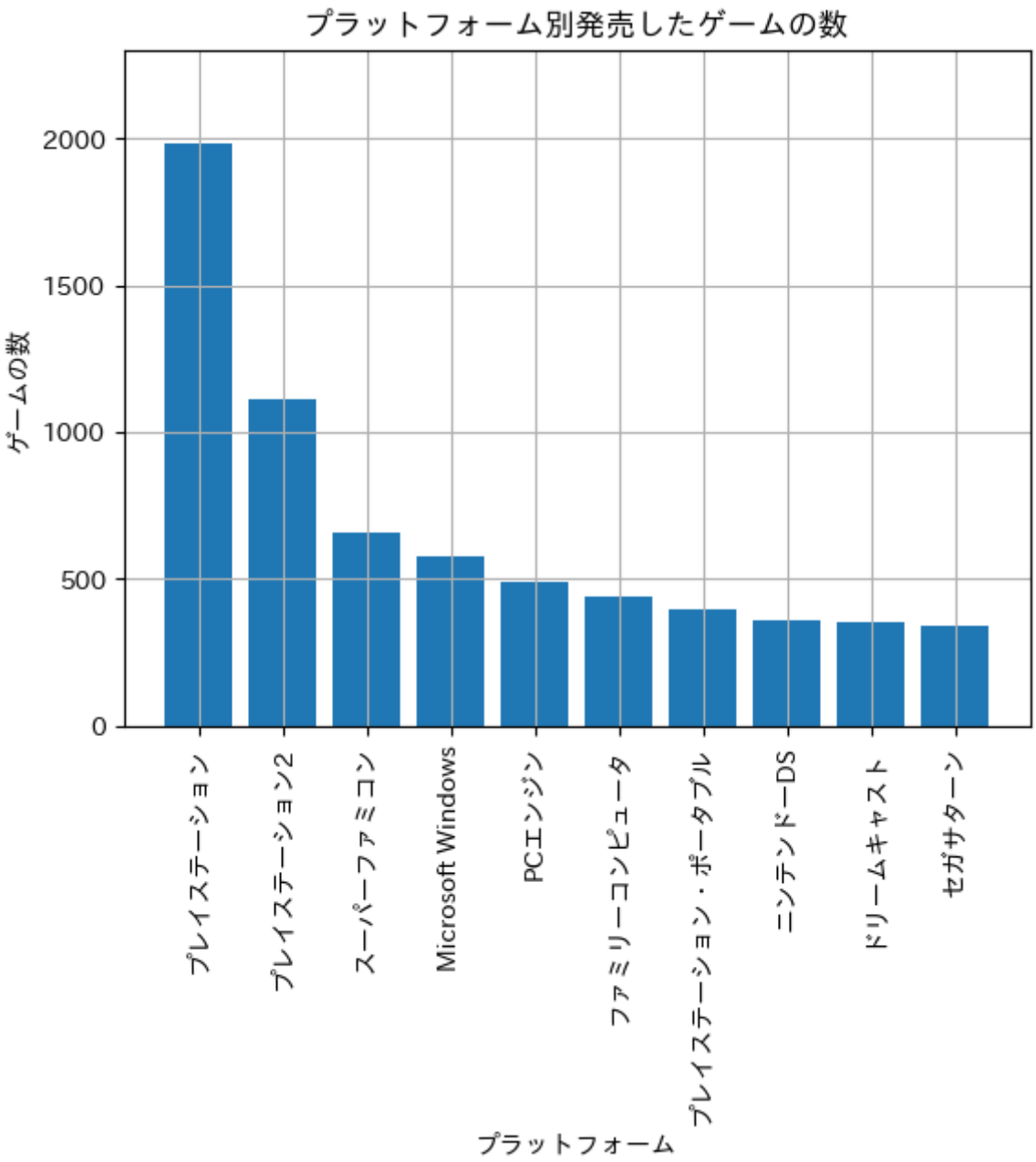


プラットフォームごとの販売されたゲームの種類数

In [ ]:
```
json_open = open('/content/metadata_gm-col_gm305_00001.json', 'r')
json_load = json.load(json_open, strict=False)
df = pd.DataFrame(json_load["@graph"])
```

In [ ]:
```
plat_df = df[['gamePlatform',"datePublished",'source']]
#プラットフォームごとに数える
plat_count=plat_df["gamePlatform"].value_counts()
#グラフ表示関数
def show_graph(x, y):
    fig, ax = plt.subplots(dpi=100)
    ax.set_title('プラットフォーム別発売したゲームの数')
    ax.set_xlabel('プラットフォーム')
    ax.set_ylabel('ゲームの数')
    ax.set_ylim(0, 2300)
    ax.grid()
    # X軸を90度回転させて縦並びにする
```

```
    plt.xticks(rotation=90)
    plt.bar(x,y)
#上位10個をプロット
show_graph(plat_count.index[0:10],plat_count[0:10])
```



折れ線グラフ

データセットの準備

以下では，メディア芸術データベースのデータを利用し，集英社が出版した漫画の種類数を時系列で可視化する． 上記のリンクから「マンガ単行本」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

※「1-4. 単回帰分析」， 「1-4. ロジスティック回帰分析」， 「1-5. 1〜3次元の図表化」，「1-5. 関係性の可視化（ネットワーク構造）」， 「1-7. ソートアルゴリズム」， 「1-7. 探索アルゴリズム」， 「2-5. データ加工」， 「3-3. 機械学習」は同じデータセットを利用するため，もし同じものを持っている場合は以下の取得作業は不要である．そちらをアップロードしよう．

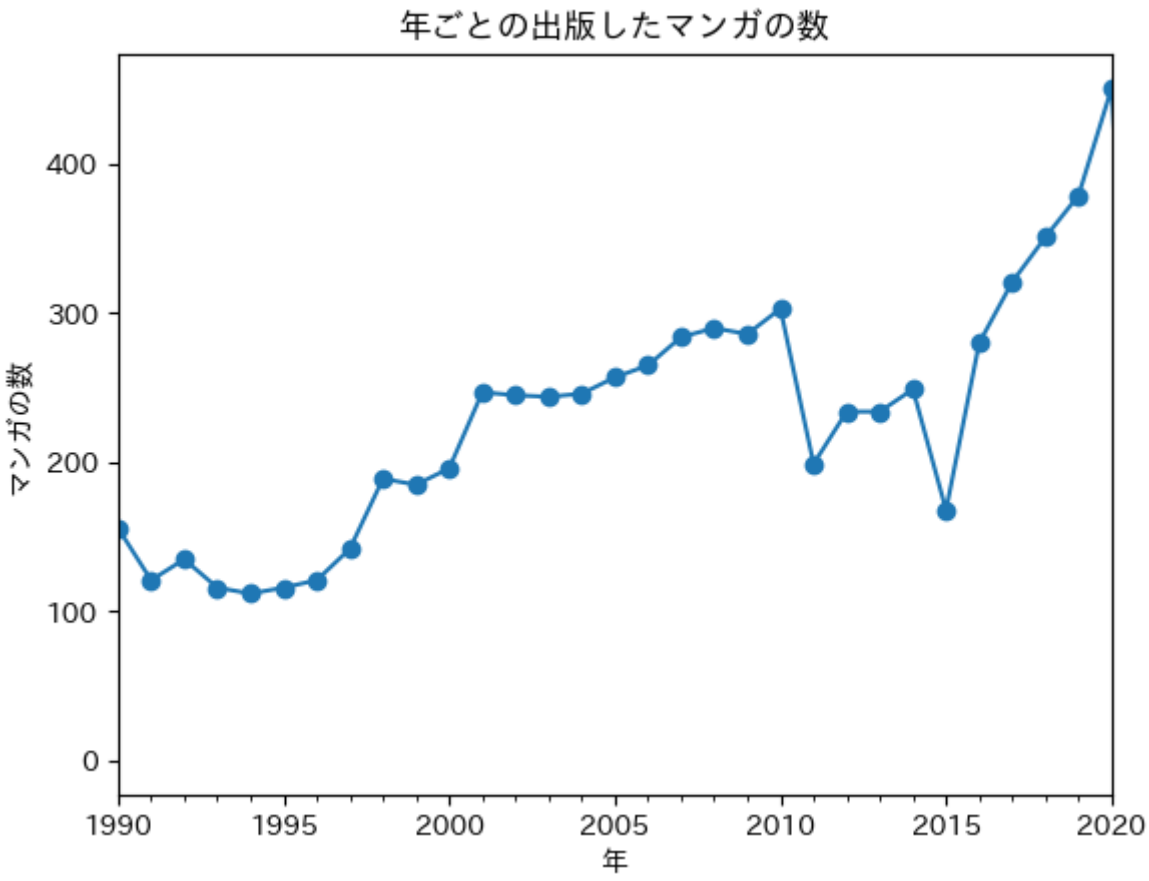
ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう．（およそ30分程度）

In [ ]:
```
json_open = open('/content/metadata_cm-item_cm101_00001.json', 'r')
json_load = json.load(json_open, strict=False)
df = pd.DataFrame(json_load["@graph"])
```

集英社が出版した年毎の漫画の種類数の推移

In [ ]:
```
#データ前処理
df = df[['publisher',"datePublished"]]
df['publisher']=df['publisher'].str[:3]
df.loc[df["datePublished"].str.len()<10,'datePublished']=df["datePublished"].astype(str)+'-01'
after_df=pd.DataFrame(df.loc[df["publisher"]=='集英社'])#時間のデータに変換
after_df['datePublished']=pd.to_datetime(after_df['datePublished'])
#年ごとに集計
year_counts = after_df.resample('Y',on='datePublished').size()

#プロット
year_counts.plot(kind='line', marker='o')
# グラフの設定
plt.title('年ごとの出版したマンガの数')
plt.xlabel('年')
plt.ylabel('マンガの数')
#プロット範囲
plt.xlim(pd.to_datetime("1990-01-01"),pd.to_datetime("2020-01-01"))
```

Out[ ]: (20.0, 50.0)

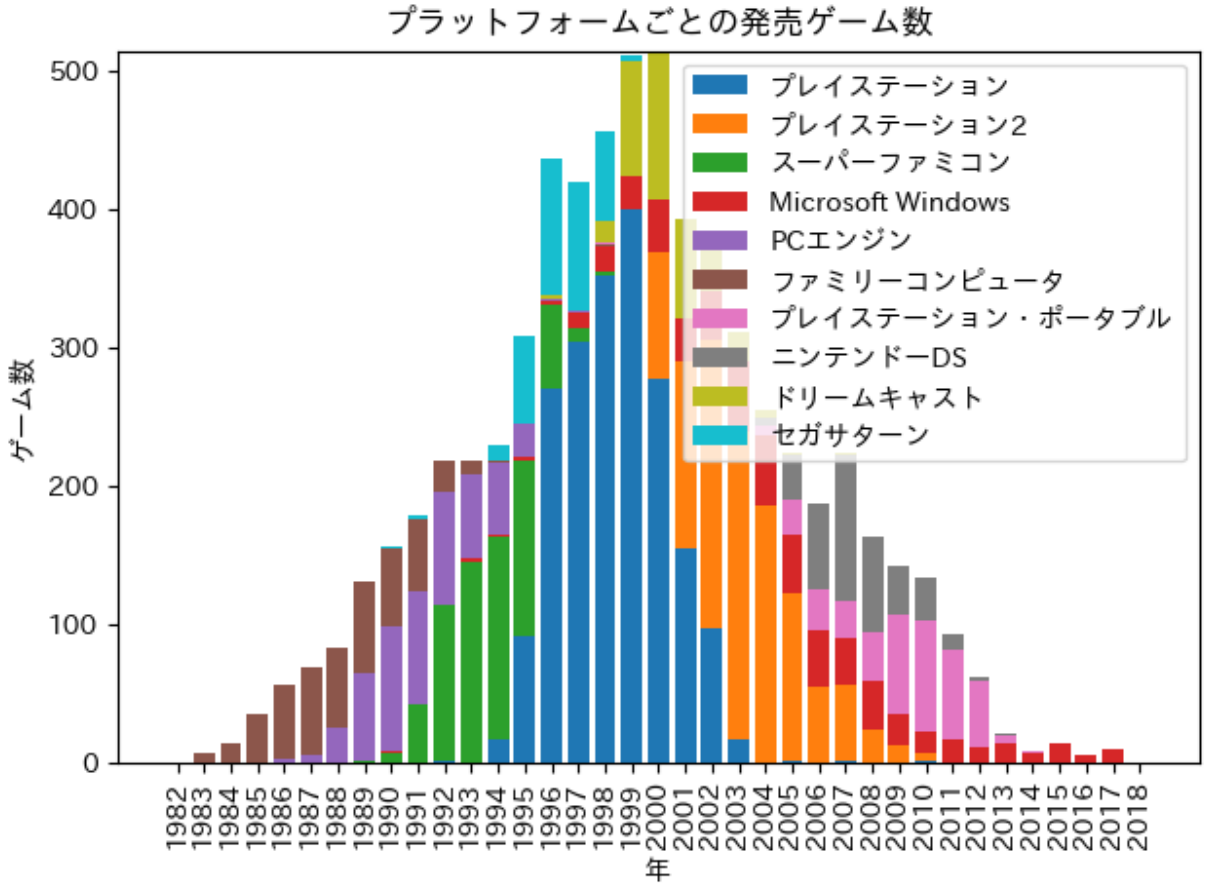


積み上げ棒グラフ

「棒グラフ」の節で扱ったデータを用いて，今度は「プラットフォーム別の発売したゲーム種類数」の時系列データを可視化する．なお，ここで可視化するデータはRCGSにて所蔵が確認されているデータのみになっているため，存在する全てのゲームパッケージデータついてのグラフではないことに注意が必要である．

「プラットフォーム別の発売したゲーム種類数」の推移

In [ ]:
```
df = plat_df.explode('gamePlatform')
yearly_counts = df.groupby(['gamePlatform', 'datePublished']).size().unstack()

# 積み上げ棒グラフのプロット
fig, ax = plt.subplots()
years = yearly_counts.columns
#積み上げるごとの中間地点
bottom = None
for platform in plat_count.index[0:10]:
    counts = yearly_counts.loc[platform].fillna(0)
    plt.xticks(rotation=90)
    #積み上げる
    ax.bar(years, counts, bottom=bottom, label=platform)
    if bottom is None:
        bottom = counts
    else:
        bottom += counts

# グラフの設定
ax.set_title('プラットフォームごとの発売ゲーム数')
ax.set_xlabel('年')
ax.set_ylabel('ゲーム数')
ax.legend()

# グラフの表示
plt.tight_layout()
plt.show()
```


プラットフォームごとの発売ゲーム数
ゲーム数
年
0
100
200
300
400
500
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
プレイステーション
プレイステーション2
スーパーファミコン
Microsoft Windows
PCエンジン
ファミリーコンピュータ
プレイステーション・ポータブル
ニンテンドーDS
ドリームキャスト
セガサターン

1-5. データの可視化 - 関係性の可視化（ネットワーク構造）

メディア芸術データベースのデータを利用して，ネットワーク構造の可視化を行なってみよう．

データセットの準備

以下では，メディア芸術データベースのデータを利用し，手塚治虫を作者に含むマンガ単行本と，出版社のネットワーク構造を可視化する． 上記のリンクから「マンガ単行本」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

※「1-4. 単回帰分析」，「1-4. ロジスティック回帰分析」，「1-5. 1〜3次元の図表化」，「1-5. 関係性の可視化（ネットワーク構造）」，「1-7. ソートアルゴリズム」，「1-7. 探索アルゴリズム」，「2-5. データ加工」，「3-3. 機械学習」は同じデータセットを利用するため，もし同じものを持っている場合は以下の取得作業は不要である．そちらをアップロードしよう．

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう． （およそ30分程度）

In [ ]:
```
import json
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
!pip install japanize-matplotlib
import japanize_matplotlib

json_open = open('/content/metadata_cm-item_cm101_00001.json', 'r')
json_load = json.load(json_open, strict=False)
```

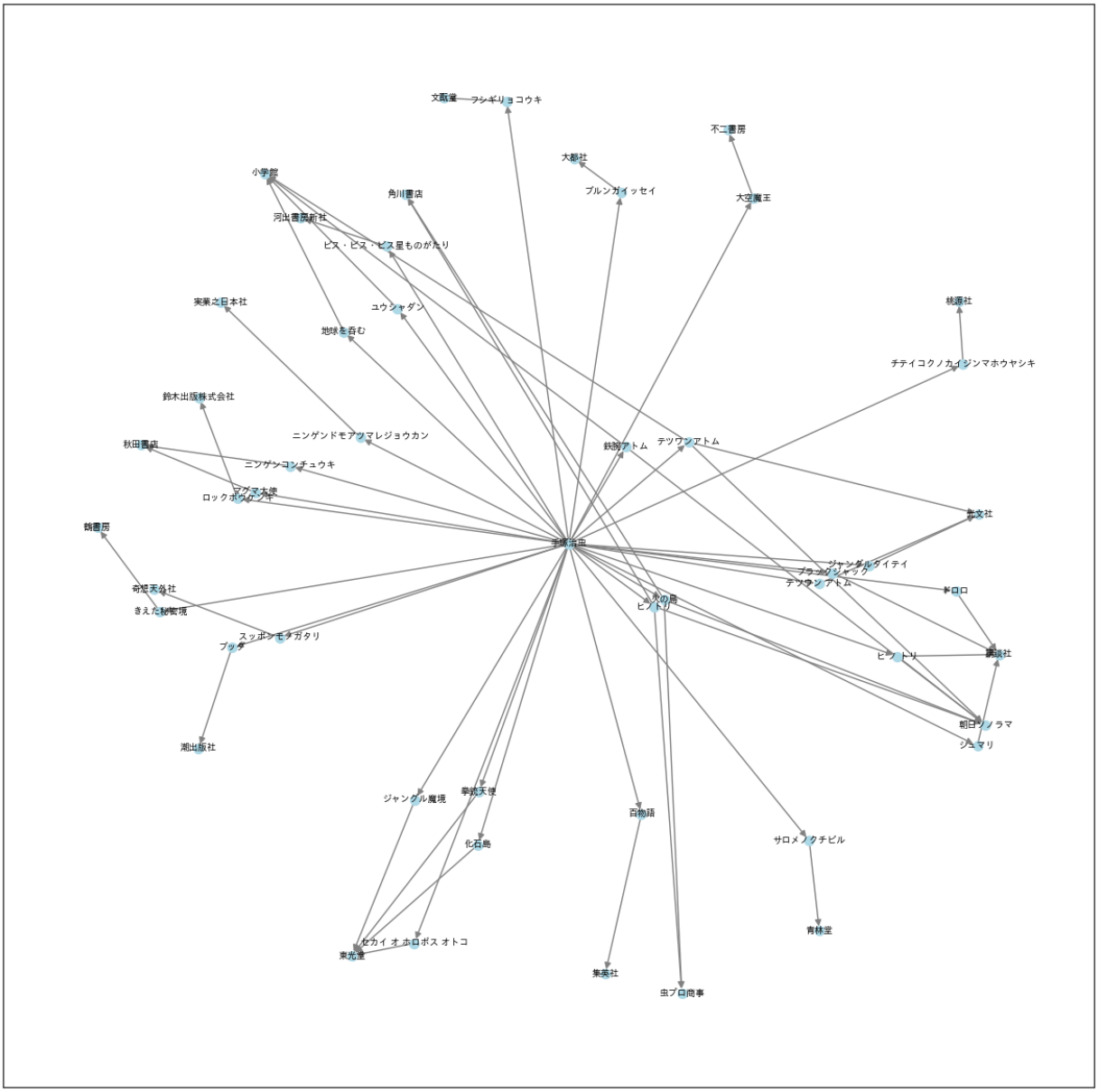
```
Requirement already satisfied: japanize-matplotlib in /usr/local/lib/python3.10/dist-packages (1.1.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from japanize-matplotlib) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (2.8.2)
```

Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->japanize-matplotlib) (1.16.0)

ネットワーク構造の可視化

In [ ]:

```
#データ前処理
df = pd.DataFrame(json_load["@graph"])
df = df[["schema:creator","publisher","name"]] # 作者名,出版社,マンガ単行本名の列に絞る

df['publisher']=df['publisher'].apply(lambda x: str(x).split(" ‖ ")[0].split(",")[0].split("]")[-1])

df['schema:creator'] = df['schema:creator'].apply(lambda x: x[0]['@value'] if isinstance(x, list) a
df['schema:creator'] = df['schema:creator'].apply(lambda x: x[0] if isinstance(x, list) and len(x) :
df['schema:creator']=df['schema:creator'].apply(lambda x: str(x).split(" ‖ ")[0].split(",")[0].sp

df['name'] = df['name'].apply(lambda x: x[0]['@value'] if isinstance(x, list) and len(x) > 0 and isi
df['name'] = df['name'].apply(lambda x: x[0] if isinstance(x, list) and len(x) > 0 else x)
df['name']=df['name'].apply(lambda x: str(x).split(" ‖ ")[0].split(",")[0].split("]")[-1])

df = df.dropna(subset=['schema:creator', 'publisher', 'name']) # 欠損値を削除
after_df = df[df['schema:creator']=='手塚治虫'] # 作者名が「手塚治虫」のデータのみに絞る

#グラフ構造を定義
G = nx.DiGraph()

# ノードの追加
nodes = set(after_df['schema:creator']) | set(after_df['name']) | set(after_df['publisher'])
G.add_nodes_from(nodes)

# エッジの追加
edges1 = zip(after_df['schema:creator'], after_df['name']) # 作者名のノードとマンガ単行本のノード
edges2 = zip(after_df['name'], after_df['publisher']) # マンガ単行本のノードと出版社のノードを繋ぐ
G.add_edges_from(edges1)
G.add_edges_from(edges2)

# ネットワークの可視化
plt.figure(figsize=(15, 15))

# ネットワークの設定
pos = nx.spring_layout(G, seed=50, k=0.3, iterations=50)  # レイアウトアルゴリズムのパラメータを調整
nx.draw_networkx(G, pos, with_labels=True, font_family='IPAexGothic', font_size=7, node_size
nx.draw_networkx_edge_labels(G, pos, edge_labels=nx.get_edge_attributes(G, 'label'), font_fa
#プロット
plt.show()
```


文藝堂
フシギリョコウキ
不二書房
大都社
ブルンガイッセイ
大空魔王
小学館
角川書店
河出書房新社
ビス・ビス・ビス星ものがたり
実業之日本社
ユウシャダン
地球を呑む
桃源社
チテイコクノカイジンマホウヤシキ
鈴木出版株式会社
ニンゲンドモアツマレジョウカン
秋田書房
鉄腕アトム
テツワンアトム
ニンゲンコンチュウキ
マグマ大使
ロックボウケンキ
集文社
手塚治虫
ジャングルタイテイ
ブラックジャック
テツワンアトム
鶏書房
奇想天外社
きえた秘密境
ドロロ
ピノトリ
スッポンモタガタリ
ブッダ
ピノトリ
講談社
朝日ソノラマ
ジュマリ
潮出版社
ジャングル魔境
季節天使
百物語
化石島
サロメノクチビル
青林堂
セカイ オホロボス オトコ
東光堂
集英社
虫プロ商事

1-7. アルゴリズム - ソートアルゴリズム（バブルソート、選択ソート、挿入ソート、マージソート）

あるデータ群を何らかの情報に沿ってある順番に並び替えることをソートという．機械的にソートを行うためには様々なアルゴリズムが考えられるため，それらがどのように実装されるのか確認し，ソートにかかる時間も見てみよう．

データセットの準備

以下では，メディア芸術データベースのデータを利用し，「ONE PIECE」のマンガタイトルを出版日順にソートしていく．上記のリンクから「マンガ単行本」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

※「1-4. 単回帰分析」，「1-4. ロジスティック回帰分析」，「1-5. 1〜3次元の図表化」，「1-5. 関係性の可視化（ネットワーク構造）」，「1-7. ソートアルゴリズム」，「1-7. 探索アルゴリズム」，「2-5. データ加工」，「3-3. 機械学習」は同じデータセットを利用するため，もし同じものを持っている場合は以下の取得作業は不要である．そちらをアップロードしよう．

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう．（およそ30分程度）

In [ ]:
```
import json
import copy
import time
```

In [ ]:
```
# ファイルを開き,読み込む
j_file=open('/content/metadata_cm-item_cm101.json')
df = json.load(j_file)

# 漫画のデータは'@graph'配下にあるので,そこを切りだす
comics=df['@graph']
```

In [ ]:
```
# ONE PIECEのタイトルを格納するリスト
onepiece=[]

# comicsデータ内の各要素(comic)を順番に見ていく
# 「タイトルに"ONE PIECE"が入っている」かつ「出版日のデータがあるもの」を"onepiece"リストに格納していく
for comic in comics:
  if('ONE PIECE' in comic['rdfs:label']):
    if('schema:datePublished' in comic.keys()):

      date=comic['schema:datePublished']
      date=date.replace('-','')
      date=date[0:6]
      onepiece.append([comic['rdfs:label'],int(date)])
```

In [ ]:
```
onepiece #before sort
```

Out[ ]:
```
[['ONE PIECE FILM STRONG WORLD 下', 201012],
 ['ONE PIECE FILM STRONG WORLD 上', 201012],
 ['ONE PIECE THE MOVIE カラクリ城のメカ巨兵', 200611],
```

['ONE PIECE THE MOVIE エピソードオブチョッパー+冬に咲く、奇跡の桜', 200911],
['ONE PIECE 巻61', 201102],
['ONE PIECE 巻60', 201011],
['ONE PIECE 巻59', 201008],
['ONE PIECE 巻58', 201006],
['ONE PIECE 巻56', 200912],
['ONE PIECE 巻55', 200909],
['ONE PIECE 巻54', 200906],
['ONE PIECE 巻53', 200903],
['ONE PIECE 巻52', 200812],
['ONE PIECE 巻51', 200809],
['ONE PIECE 巻50', 200806],
['ONE PIECE 巻49', 200803],
['ONE PIECE 巻48', 200712],
['ONE PIECE 巻47', 200709],
['ONE PIECE 巻45', 200703],
['ONE PIECE 巻46', 200707],
['ONE PIECE 巻57', 201003],
['ONE PIECE 巻44', 200612],
['ONE PIECE 巻43', 200609],
['ONE PIECE 巻42', 200607],
['ONE PIECE 巻41', 200604],
['ONE PIECE 巻40', 200512],
['ONE PIECE 巻39', 200511],
['ONE PIECE 巻38', 200507],
['ONE PIECE 巻36', 200502],
['ONE PIECE 巻35', 200411],
['ONE PIECE 巻34', 200408],
['ONE PIECE 巻33', 200406],
['ONE PIECE 巻32', 200403],
['ONE PIECE 巻31', 200312],
['ONE PIECE 巻30', 200310],
['ONE PIECE 巻28', 200305],
['ONE PIECE 巻27', 200302],
['ONE PIECE 巻26', 200212],
['ONE PIECE 巻25', 200209],
['ONE PIECE 巻24', 200207],
['ONE PIECE 巻23', 200204],
['ONE PIECE 巻37', 200505],
['ONE PIECE THE 2ND LOG “SANJI', 200511],
['ONE PIECE THE 14TH LOG “FRANKY', 200908],
['ONE PIECE THE 13TH LOG “NICO ROBIN', 200907],
['ONE PIECE THE 12TH LOG “ROCKET MAN', 200906],
['ONE PIECE THE 11TH LOG “WATER SEVEN', 200904],
['ONE PIECE THE 9TH LOG “GOD', 201003],
['ONE PIECE THE 8TH LOG “SKYPIEA', 201003],
['ONE PIECE THE 7TH LOG “VIVI', 201002],
['ONE PIECE THE 7TH LOG “VIVI', 200605],
['ONE PIECE THE 6TH LOG “ARABASTA', 201002],
['ONE PIECE THE 6TH LOG AR“ABASTA', 200603],
['ONE PIECE THE 5TH LOG “CHOPPER', 201001],
['ONE PIECE THE 5TH LOG “CHOPPER', 200602],
['ONE PIECE THE 4TH LOG “GRAND LINE', 201001],
['ONE PIECE THE 4TH LOG “GRAND LINE', 200601],
['ONE PIECE THE 3RD LOG “NAMI', 200512],
['ONE PIECE THE FIRST LOG', 200504],
['劇場版 ONE PIECE', 200803],
['ONE PIECE THE 10TH LOG “BELL', 200806],

['ONE PIECE EXTRA LOG 1', 201212],
['ONE PIECE FILM Z 上', 201307],
['ONE PIECE FILM Z 下', 201307],
['ONE PIECE 巻63', 201108],
['ONE PIECE 巻64', 201111],
['ONE PIECE 巻65', 201202],
['ONE PIECE 巻66', 201205],
['ONE PIECE 巻67', 201208],
['ONE PIECE 巻68', 201211],
['ONE PIECE 巻69', 201303],
['ONE PIECE 巻70', 201306],
['ONE PIECE 巻71', 201308],
['ONE PIECE 総集編 THE11TH LOG', 201208],
['ONE PIECE 総集編 THE12TH LOG', 201208],
['ONE PIECE 総集編 THE13TH LOG', 201208],
['ONE PIECE 総集編 THE14TH LOG', 201208],
['ONE PIECE 総集編 THE15TH LOG', 201208],
['ONE PIECE 総集編 THE16TH LOG', 201208],
['ONE PIECE 総集編 THE17TH LOG', 201209],
['ONE PIECE 総集編 THE18TH LOG', 201210],
['ONE PIECE 総集編 THE19TH LOG', 201211],
['ONE PIECE 総集編 THE20TH LOG', 201212],
['ONE PIECE THE MOVIEデッドエンドの冒険', 201102],
['ONE PIECE呪われた聖剣', 201102],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻3', 201907],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻2', 201907],
['ONE PIECE 巻93', 201907],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻1", 201907],
['ONE PIECE 巻94', 201910],
['ONE PIECE 巻95', 201912],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 上', 202005],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 下', 202005],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻2", 202004],
['ワンピースパーティー = ONE PIECE PARTY 6', 202004],
['ONE PIECE 巻96', 202004],
['ONE PIECE学園 1', 202004],
['ONE PIECE 巻97', 202009],
['ONE PIECE 巻98', 202102],
['ONE PIECE 巻99', 202106],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻3", 202108],
['ONE PIECE 巻100', 202109],
['ONE PIECE学園 3', 202112],
['ONE PIECE 巻101', 202112],
['ONE PIECE 巻102', 202204],
['ONE PIECE学園 4', 202204],
['ONE PIECEエピソードエース 1', 202208],
['ONE PIECE 巻103', 202208],
['ONE PIECEエピソードエース 2', 202208],
['ONE PIECE学園 5', 202211],
['ONE PIECE 巻104', 202211],
['ONE PIECE学園 6', 202303],
['ONE PIECE 巻105', 202303],
['ONE PIECE 巻62', 201105],
['ONE PIECE THE 15TH LOG“THRILLER BARK”', 201102],
['ONE PIECE THE 16TH LOG“THRILLER BROOK”', 201102],
['ONE PIECE 巻72', 201311],
['ONE PIECE COLOR WALK 6', 201401],
['ONE PIECE 巻73', 201403],

['ONE PIECE 巻74', 201406],
['ONE PIECE 巻75', 201409],
['ファンが選ぶONE PIECE“涙”ベスト10!! 1', 201412],
['ファンが選ぶONE PIECE”涙”ベスト10!! II', 201412],
['ONE PIECE 巻77', 201504],
['ONE PIECE 巻78', 201507],
['ONE PIECE 巻76', 201412],
['ONE PIECE 巻79', 201510],
['ONE PIECE 巻80', 201512],
['ONE PIECE 巻81', 201604],
['ONE PIECE 巻82', 201607],
['ONE PIECE COLOR WALK 7', 201607],
['ONE PIECE 巻83', 201611],
['ONE PIECE 巻84', 201702],
['ONE PIECE 巻85', 201705],
['ONE PIECE THE 21ST LOG“2 YEARS LATER”', 201706],
['ONE PIECE FILM GOLD 上', 201707],
['ONE PIECE FILM GOLD 下', 201707],
['ONE PIECE 総集編 THE 22ND LOG', 201707],
['ONE PIECE 巻86', 201708],
['ONE PIECE 総集編 THE 23RD LOG', 201708],
['ONE PIECE 総集編 THE 24TH LOG', 201709],
['ONE PIECE総集編 THE 25TH LOG', 201710],
['ONE PIECE 巻87', 201711],
['ONE PIECE総集編 THE 26TH LOG', 201711],
['ONE PIECE総集編 THE 27TH LOG', 201712],
['ONE PIECE総集編 THE 28TH LOG', 201801],
['ONE PIECE 巻88', 201803],
['ONE PIECE 巻91', 201812],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻1', 201812],
['ONE PIECE 巻92', 201903],
['ワンピースパーティー = ONE PIECE PARTY 5', 201903],
['ONE PIECE 巻90', 201809],
['ONE PIECE 巻89', 201806],
['ワンピースパーティー = ONE PIECE PARTY 4', 201806],
['ONE PIECE COLOR WALK 5', 201012],
['ONE PIECE COLOR WALK 4', 201003],
['ONE PIECE COLOR WALK 3', 200601],
['ONE PIECE 巻22', 200202],
['ONE PIECE 巻21', 200112],
['ONE PIECE 巻20', 200109],
['ONE PIECE 巻19', 200107],
['ONE PIECE 巻18', 200104],
['ONE PIECE 巻17', 200102],
['ONE PIECE 巻16', 200012],
['ONE PIECE 巻6', 199812],
['ONE PIECE 巻4', 199808],
['ONE PIECE 巻5', 199810],
['ONE PIECE 巻3', 199806],
['ONE PIECE 巻2', 199804],
['ONE PIECE 巻1', 199712],
['ONE PIECE COLOR WALK 2', 200311],
['ONE PIECE RED', 200201],
['ONE PIECE 巻29', 200307],
['ONE PIECE 巻15', 200009],
['ONE PIECE 巻14', 200007],
['ONE PIECE 巻13', 200005],
['ONE PIECE 巻12', 200002],

['ONE PIECE 巻11', 199912],
['ONE PIECE 巻10', 199910],
['ONE PIECE 巻9', 199907],
['ONE PIECE 巻8', 199905],
['ONE PIECE 巻7', 199903]]

ソート

バブルソート

In [ ]:
```
# onepieceリストのコピーを作り,それをソートする
copy_onepiece = copy.deepcopy(onepiece)

# 時間計測開始
start = time.time()

# 要素を頭から順番に見ていく
for i in range(len(copy_onepiece)-1):
  # copy_onepiece[0]~[終端]の中で,値の大きいものを次の値と交換するということを繰り返し,最大値を持つ
  # 終端側(値の大きい方)のi個がソート済みになるので,新しく最大値が終端に追いやられるたびに終端の位置
  for j in range(len(copy_onepiece)-1-i):
    if copy_onepiece[j][1]>copy_onepiece[j+1][1]:
      tmp1=copy_onepiece[j]
      tmp2=copy_onepiece[j+1]
      copy_onepiece[j]=tmp2
      copy_onepiece[j+1]=tmp1

# 時間計測終了
print("計算時間: ", time.time() - start)

copy_onepiece
```

計算時間: 0.0038075447082519531

Out[ ]:
```
[['ONE PIECE 巻1', 199712],
 ['ONE PIECE 巻2', 199804],
 ['ONE PIECE 巻3', 199806],
 ['ONE PIECE 巻4', 199808],
 ['ONE PIECE 巻5', 199810],
 ['ONE PIECE 巻6', 199812],
 ['ONE PIECE 巻7', 199903],
 ['ONE PIECE 巻8', 199905],
 ['ONE PIECE 巻9', 199907],
 ['ONE PIECE 巻10', 199910],
 ['ONE PIECE 巻11', 199912],
 ['ONE PIECE 巻12', 200002],
 ['ONE PIECE 巻13', 200005],
 ['ONE PIECE 巻14', 200007],
 ['ONE PIECE 巻15', 200009],
 ['ONE PIECE 巻16', 200012],
 ['ONE PIECE 巻17', 200102],
 ['ONE PIECE 巻18', 200104],
 ['ONE PIECE 巻19', 200107],
 ['ONE PIECE 巻20', 200109],
 ['ONE PIECE 巻21', 200112],
 ['ONE PIECE RED', 200201],
 ['ONE PIECE 巻22', 200202],
 ['ONE PIECE 巻23', 200204],
 ['ONE PIECE 巻24', 200207],
 ['ONE PIECE 巻25', 200209],
```

['ONE PIECE 巻26', 200212],
['ONE PIECE 巻27', 200302],
['ONE PIECE 巻28', 200305],
['ONE PIECE 巻29', 200307],
['ONE PIECE 巻30', 200310],
['ONE PIECE COLOR WALK 2', 200311],
['ONE PIECE 巻31', 200312],
['ONE PIECE 巻32', 200403],
['ONE PIECE 巻33', 200406],
['ONE PIECE 巻34', 200408],
['ONE PIECE 巻35', 200411],
['ONE PIECE 巻36', 200502],
['ONE PIECE THE FIRST LOG', 200504],
['ONE PIECE 巻37', 200505],
['ONE PIECE 巻38', 200507],
['ONE PIECE 巻39', 200511],
['ONE PIECE THE 2ND LOG “SANJI', 200511],
['ONE PIECE 巻40', 200512],
['ONE PIECE THE 3RD LOG “NAMI', 200512],
['ONE PIECE THE 4TH LOG “GRAND LINE', 200601],
['ONE PIECE COLOR WALK 3', 200601],
['ONE PIECE THE 5TH LOG “CHOPPER', 200602],
['ONE PIECE THE 6TH LOG AR“ABASTA', 200603],
['ONE PIECE 巻41', 200604],
['ONE PIECE THE 7TH LOG “VIVI', 200605],
['ONE PIECE 巻42', 200607],
['ONE PIECE 巻43', 200609],
['ONE PIECE THE MOVIE カラクリ城のメカ巨兵', 200611],
['ONE PIECE 巻44', 200612],
['ONE PIECE 巻45', 200703],
['ONE PIECE 巻46', 200707],
['ONE PIECE 巻47', 200709],
['ONE PIECE 巻48', 200712],
['ONE PIECE 巻49', 200803],
['劇場版 ONE PIECE', 200803],
['ONE PIECE 巻50', 200806],
['ONE PIECE THE 10TH LOG “BELL', 200806],
['ONE PIECE 巻51', 200809],
['ONE PIECE 巻52', 200812],
['ONE PIECE 巻53', 200903],
['ONE PIECE THE 11TH LOG “WATER SEVEN', 200904],
['ONE PIECE 巻54', 200906],
['ONE PIECE THE 12TH LOG “ROCKET MAN', 200906],
['ONE PIECE THE 13TH LOG “NICO ROBIN', 200907],
['ONE PIECE THE 14TH LOG “FRANKY', 200908],
['ONE PIECE 巻55', 200909],
['ONE PIECE THE MOVIE エピソードオブチョッパー+冬に咲く、奇跡の桜', 200911],
['ONE PIECE 巻56', 200912],
['ONE PIECE THE 5TH LOG “CHOPPER', 201001],
['ONE PIECE THE 4TH LOG “GRAND LINE', 201001],
['ONE PIECE THE 7TH LOG “VIVI', 201002],
['ONE PIECE THE 6TH LOG “ARABASTA', 201002],
['ONE PIECE 巻57', 201003],
['ONE PIECE THE 9TH LOG “GOD', 201003],
['ONE PIECE THE 8TH LOG “SKYPIEA', 201003],
['ONE PIECE COLOR WALK 4', 201003],
['ONE PIECE 巻58', 201006],
['ONE PIECE 巻59', 201008],

['ONE PIECE 巻60', 201011],
['ONE PIECE FILM STRONG WORLD 下', 201012],
['ONE PIECE FILM STRONG WORLD 上', 201012],
['ONE PIECE COLOR WALK 5', 201012],
['ONE PIECE 巻61', 201102],
['ONE PIECE THE MOVIEデッドエンドの冒険', 201102],
['ONE PIECE呪われた聖剣', 201102],
['ONE PIECE THE 15TH LOG“THRILLER BARK”', 201102],
['ONE PIECE THE 16TH LOG“THRILLER BROOK”', 201102],
['ONE PIECE 巻62', 201105],
['ONE PIECE 巻63', 201108],
['ONE PIECE 巻64', 201111],
['ONE PIECE 巻65', 201202],
['ONE PIECE 巻66', 201205],
['ONE PIECE 巻67', 201208],
['ONE PIECE 総集編 THE11TH LOG', 201208],
['ONE PIECE 総集編 THE12TH LOG', 201208],
['ONE PIECE 総集編 THE13TH LOG', 201208],
['ONE PIECE 総集編 THE14TH LOG', 201208],
['ONE PIECE 総集編 THE15TH LOG', 201208],
['ONE PIECE 総集編 THE16TH LOG', 201208],
['ONE PIECE 総集編 THE17TH LOG', 201209],
['ONE PIECE 総集編 THE18TH LOG', 201210],
['ONE PIECE 巻68', 201211],
['ONE PIECE 総集編 THE19TH LOG', 201211],
['ONE PIECE EXTRA LOG 1', 201212],
['ONE PIECE 総集編 THE20TH LOG', 201212],
['ONE PIECE 巻69', 201303],
['ONE PIECE 巻70', 201306],
['ONE PIECE FILM Z 上', 201307],
['ONE PIECE FILM Z 下', 201307],
['ONE PIECE 巻71', 201308],
['ONE PIECE 巻72', 201311],
['ONE PIECE COLOR WALK 6', 201401],
['ONE PIECE 巻73', 201403],
['ONE PIECE 巻74', 201406],
['ONE PIECE 巻75', 201409],
['ファンが選ぶONE PIECE“涙”ベスト10!! 1', 201412],
['ファンが選ぶONE PIECE”涙”ベスト10!! II', 201412],
['ONE PIECE 巻76', 201412],
['ONE PIECE 巻77', 201504],
['ONE PIECE 巻78', 201507],
['ONE PIECE 巻79', 201510],
['ONE PIECE 巻80', 201512],
['ONE PIECE 巻81', 201604],
['ONE PIECE 巻82', 201607],
['ONE PIECE COLOR WALK 7', 201607],
['ONE PIECE 巻83', 201611],
['ONE PIECE 巻84', 201702],
['ONE PIECE 巻85', 201705],
['ONE PIECE THE 21ST LOG“2 YEARS LATER”', 201706],
['ONE PIECE FILM GOLD 上', 201707],
['ONE PIECE FILM GOLD 下', 201707],
['ONE PIECE 総集編 THE 22ND LOG', 201707],
['ONE PIECE 巻86', 201708],
['ONE PIECE 総集編 THE 23RD LOG', 201708],
['ONE PIECE 総集編 THE 24TH LOG', 201709],
['ONE PIECE総集編 THE 25TH LOG', 201710],

```
['ONE PIECE 巻87', 201711],
['ONE PIECE総集編 THE 26TH LOG', 201711],
['ONE PIECE総集編 THE 27TH LOG', 201712],
['ONE PIECE総集編 THE 28TH LOG', 201801],
['ONE PIECE 巻88', 201803],
['ONE PIECE 巻89', 201806],
['ワンピースパーティー = ONE PIECE PARTY 4', 201806],
['ONE PIECE 巻90', 201809],
['ONE PIECE 巻91', 201812],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻1', 201812],
['ONE PIECE 巻92', 201903],
['ワンピースパーティー = ONE PIECE PARTY 5', 201903],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻3', 201907],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻2', 201907],
['ONE PIECE 巻93', 201907],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻1", 201907],
['ONE PIECE 巻94', 201910],
['ONE PIECE 巻95', 201912],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻2", 202004],
['ワンピースパーティー = ONE PIECE PARTY 6', 202004],
['ONE PIECE 巻96', 202004],
['ONE PIECE学園 1', 202004],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 上', 202005],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 下', 202005],
['ONE PIECE 巻97', 202009],
['ONE PIECE 巻98', 202102],
['ONE PIECE 巻99', 202106],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻3", 202108],
['ONE PIECE 巻100', 202109],
['ONE PIECE学園 3', 202112],
['ONE PIECE 巻101', 202112],
['ONE PIECE 巻102', 202204],
['ONE PIECE学園 4', 202204],
['ONE PIECEエピソードエース 1', 202208],
['ONE PIECE 巻103', 202208],
['ONE PIECEエピソードエース 2', 202208],
['ONE PIECE学園 5', 202211],
['ONE PIECE 巻104', 202211],
['ONE PIECE学園 6', 202303],
['ONE PIECE 巻105', 202303]]
```

選択ソート

In [ ]:

```
# onepieceリストのコピーを作り,それをソートする
copy_onepiece = copy.deepcopy(onepiece)

# 時間計測開始
start = time.time()

# 頭から順番に要素を見ていく
for i in range(len(copy_onepiece)-1):
  # 今見ている要素copy_onepiece[i]の出版日をとりあえずの最小値としておく
  tmp=copy_onepiece[i]
  min=copy_onepiece[i][1]
  index=i
  # copy_onepiece[i]よりも後の全ての要素(copy_onepiece[j],...)を順番に見ていく
  # 今最小値としているものよりも小さい値を持つデータがあれば,それを最小値として覚え直す
  for j in range(i+1,len(copy_onepiece)):
    if copy_onepiece[j][1]<min:
```

```
            min=copy_onepiece[j][1]
            index=j
        # 最小値を持つデータとi番目のデータと交換する
        copy_onepiece[i]=copy_onepiece[index]
        copy_onepiece[index]=tmp

    # 時間計測終了
    print("計算時間: ", time.time() - start)

    copy_onepiece
```

```
計算時間:  0.0013833045959472656
```

Out[ ]:
```
[['ONE PIECE 巻1', 199712],
 ['ONE PIECE 巻2', 199804],
 ['ONE PIECE 巻3', 199806],
 ['ONE PIECE 巻4', 199808],
 ['ONE PIECE 巻5', 199810],
 ['ONE PIECE 巻6', 199812],
 ['ONE PIECE 巻7', 199903],
 ['ONE PIECE 巻8', 199905],
 ['ONE PIECE 巻9', 199907],
 ['ONE PIECE 巻10', 199910],
 ['ONE PIECE 巻11', 199912],
 ['ONE PIECE 巻12', 200002],
 ['ONE PIECE 巻13', 200005],
 ['ONE PIECE 巻14', 200007],
 ['ONE PIECE 巻15', 200009],
 ['ONE PIECE 巻16', 200012],
 ['ONE PIECE 巻17', 200102],
 ['ONE PIECE 巻18', 200104],
 ['ONE PIECE 巻19', 200107],
 ['ONE PIECE 巻20', 200109],
 ['ONE PIECE 巻21', 200112],
 ['ONE PIECE RED', 200201],
 ['ONE PIECE 巻22', 200202],
 ['ONE PIECE 巻23', 200204],
 ['ONE PIECE 巻24', 200207],
 ['ONE PIECE 巻25', 200209],
 ['ONE PIECE 巻26', 200212],
 ['ONE PIECE 巻27', 200302],
 ['ONE PIECE 巻28', 200305],
 ['ONE PIECE 巻29', 200307],
 ['ONE PIECE 巻30', 200310],
 ['ONE PIECE COLOR WALK 2', 200311],
 ['ONE PIECE 巻31', 200312],
 ['ONE PIECE 巻32', 200403],
 ['ONE PIECE 巻33', 200406],
 ['ONE PIECE 巻34', 200408],
 ['ONE PIECE 巻35', 200411],
 ['ONE PIECE 巻36', 200502],
 ['ONE PIECE THE FIRST LOG', 200504],
 ['ONE PIECE 巻37', 200505],
 ['ONE PIECE 巻38', 200507],
 ['ONE PIECE THE 2ND LOG “SANJI', 200511],
 ['ONE PIECE 巻39', 200511],
 ['ONE PIECE THE 3RD LOG “NAMI', 200512],
 ['ONE PIECE 巻40', 200512],
 ['ONE PIECE THE 4TH LOG “GRAND LINE', 200601],
 ['ONE PIECE COLOR WALK 3', 200601],
```

['ONE PIECE THE 5TH LOG “CHOPPER', 200602],
['ONE PIECE THE 6TH LOG AR“ABASTA', 200603],
['ONE PIECE 巻41', 200604],
['ONE PIECE THE 7TH LOG “VIVI', 200605],
['ONE PIECE 巻42', 200607],
['ONE PIECE 巻43', 200609],
['ONE PIECE THE MOVIE カラクリ城のメカ巨兵', 200611],
['ONE PIECE 巻44', 200612],
['ONE PIECE 巻45', 200703],
['ONE PIECE 巻46', 200707],
['ONE PIECE 巻47', 200709],
['ONE PIECE 巻48', 200712],
['劇場版 ONE PIECE', 200803],
['ONE PIECE 巻49', 200803],
['ONE PIECE THE 10TH LOG “BELL', 200806],
['ONE PIECE 巻50', 200806],
['ONE PIECE 巻51', 200809],
['ONE PIECE 巻52', 200812],
['ONE PIECE 巻53', 200903],
['ONE PIECE THE 11TH LOG “WATER SEVEN', 200904],
['ONE PIECE THE 12TH LOG “ROCKET MAN', 200906],
['ONE PIECE 巻54', 200906],
['ONE PIECE THE 13TH LOG “NICO ROBIN', 200907],
['ONE PIECE THE 14TH LOG “FRANKY', 200908],
['ONE PIECE 巻55', 200909],
['ONE PIECE THE MOVIE エピソードオブチョッパー+冬に咲く、奇跡の桜', 200911],
['ONE PIECE 巻56', 200912],
['ONE PIECE THE 4TH LOG “GRAND LINE', 201001],
['ONE PIECE THE 5TH LOG “CHOPPER', 201001],
['ONE PIECE THE 7TH LOG “VIVI', 201002],
['ONE PIECE THE 6TH LOG “ARABASTA', 201002],
['ONE PIECE COLOR WALK 4', 201003],
['ONE PIECE THE 8TH LOG “SKYPIEA', 201003],
['ONE PIECE 巻57', 201003],
['ONE PIECE THE 9TH LOG “GOD', 201003],
['ONE PIECE 巻58', 201006],
['ONE PIECE 巻59', 201008],
['ONE PIECE 巻60', 201011],
['ONE PIECE COLOR WALK 5', 201012],
['ONE PIECE FILM STRONG WORLD 上', 201012],
['ONE PIECE FILM STRONG WORLD 下', 201012],
['ONE PIECE THE 15TH LOG“THRILLER BARK”', 201102],
['ONE PIECE THE 16TH LOG“THRILLER BROOK”', 201102],
['ONE PIECE呪われた聖剣', 201102],
['ONE PIECE 巻61', 201102],
['ONE PIECE THE MOVIEデッドエンドの冒険', 201102],
['ONE PIECE 巻62', 201105],
['ONE PIECE 巻63', 201108],
['ONE PIECE 巻64', 201111],
['ONE PIECE 巻65', 201202],
['ONE PIECE 巻66', 201205],
['ONE PIECE 総集編 THE16TH LOG', 201208],
['ONE PIECE 総集編 THE12TH LOG', 201208],
['ONE PIECE 総集編 THE13TH LOG', 201208],
['ONE PIECE 総集編 THE14TH LOG', 201208],
['ONE PIECE 総集編 THE15TH LOG', 201208],
['ONE PIECE 巻67', 201208],
['ONE PIECE 総集編 THE11TH LOG', 201208],

['ONE PIECE 総集編 THE17TH LOG', 201209],
['ONE PIECE 総集編 THE18TH LOG', 201210],
['ONE PIECE 巻68', 201211],
['ONE PIECE 総集編 THE19TH LOG', 201211],
['ONE PIECE EXTRA LOG 1', 201212],
['ONE PIECE 総集編 THE20TH LOG', 201212],
['ONE PIECE 巻69', 201303],
['ONE PIECE 巻70', 201306],
['ONE PIECE FILM Z 上', 201307],
['ONE PIECE FILM Z 下', 201307],
['ONE PIECE 巻71', 201308],
['ONE PIECE 巻72', 201311],
['ONE PIECE COLOR WALK 6', 201401],
['ONE PIECE 巻73', 201403],
['ONE PIECE 巻74', 201406],
['ONE PIECE 巻75', 201409],
['ファンが選ぶONE PIECE“涙”ベスト10!! 1', 201412],
['ファンが選ぶONE PIECE”涙”ベスト10!! II', 201412],
['ONE PIECE 巻76', 201412],
['ONE PIECE 巻77', 201504],
['ONE PIECE 巻78', 201507],
['ONE PIECE 巻79', 201510],
['ONE PIECE 巻80', 201512],
['ONE PIECE 巻81', 201604],
['ONE PIECE 巻82', 201607],
['ONE PIECE COLOR WALK 7', 201607],
['ONE PIECE 巻83', 201611],
['ONE PIECE 巻84', 201702],
['ONE PIECE 巻85', 201705],
['ONE PIECE THE 21ST LOG“2 YEARS LATER”', 201706],
['ONE PIECE FILM GOLD 上', 201707],
['ONE PIECE FILM GOLD 下', 201707],
['ONE PIECE 総集編 THE 22ND LOG', 201707],
['ONE PIECE 巻86', 201708],
['ONE PIECE 総集編 THE 23RD LOG', 201708],
['ONE PIECE 総集編 THE 24TH LOG', 201709],
['ONE PIECE総集編 THE 25TH LOG', 201710],
['ONE PIECE 巻87', 201711],
['ONE PIECE総集編 THE 26TH LOG', 201711],
['ONE PIECE総集編 THE 27TH LOG', 201712],
['ONE PIECE総集編 THE 28TH LOG', 201801],
['ONE PIECE 巻88', 201803],
['ONE PIECE 巻89', 201806],
['ワンピースパーティー = ONE PIECE PARTY 4', 201806],
['ONE PIECE 巻90', 201809],
['ONE PIECE 巻91', 201812],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻1', 201812],
['ワンピースパーティー = ONE PIECE PARTY 5', 201903],
['ONE PIECE 巻92', 201903],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻3', 201907],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻2', 201907],
['ONE PIECE 巻93', 201907],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻1", 201907],
['ONE PIECE 巻94', 201910],
['ONE PIECE 巻95', 201912],
['ONE PIECE学園 1', 202004],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻2", 202004],
['ワンピースパーティー = ONE PIECE PARTY 6', 202004],

['ONE PIECE 巻96', 202004],
['劇場版ONE PIECE STAMPEDE : アニメコミックス 上', 202005],
['劇場版ONE PIECE STAMPEDE : アニメコミックス 下', 202005],
['ONE PIECE 巻97', 202009],
['ONE PIECE 巻98', 202102],
['ONE PIECE 巻99', 202106],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻3", 202108],
['ONE PIECE 巻100', 202109],
['ONE PIECE学園 3', 202112],
['ONE PIECE 巻101', 202112],
['ONE PIECE学園 4', 202204],
['ONE PIECE 巻102', 202204],
['ONE PIECE 巻103', 202208],
['ONE PIECEエピソードエース 2', 202208],
['ONE PIECEエピソードエース 1', 202208],
['ONE PIECE 巻104', 202211],
['ONE PIECE学園 5', 202211],
['ONE PIECE 巻105', 202303],
['ONE PIECE学園 6', 202303]]

挿入ソート

In [ ]:
```
# 新しいリストを作り,そこにソートしたものが入っていく
sorted_onepiece = []

# 時間計測開始
start = time.time()

# 先頭の次の要素から順番に見ていく
for i in range(len(onepiece)):
  flag=True
  tmp=onepiece[i][1]
  # 一番最初の操作は,onepieceの最初の要素をsorted_onepieceにコピーするだけ
  if i ==0:
    sorted_onepiece.append(onepiece[i])
    flag=False

  index=0
  j=0

  # 2巡目以降の操作では,sorted_onpiece(ソート済みのonepiece[0]~[i-1]のリスト)を先頭から見ていき,今
  while flag:
    # 最後まで見切ってしまった時の終了条件
    if j==i+1:
      index=j
      flag=False
    # 挿入するべき位置が見つかった時の終了条件
    elif sorted_onepiece[j][1]>tmp:
      index=j
      flag=False
    # 比較する位置をずらす
    j=j+1
  sorted_onepiece.insert(index,onepiece[i])

# 時間計測終了
print("計算時間: ", time.time() - start)

sorted_onepiece
```

計算時間: 0.0015192031860351562

Out[ ]: [['ONE PIECE 巻1', 199712],
['ONE PIECE 巻2', 199804],
['ONE PIECE 巻3', 199806],
['ONE PIECE 巻4', 199808],
['ONE PIECE 巻5', 199810],
['ONE PIECE 巻6', 199812],
['ONE PIECE 巻7', 199903],
['ONE PIECE 巻8', 199905],
['ONE PIECE 巻9', 199907],
['ONE PIECE 巻10', 199910],
['ONE PIECE 巻11', 199912],
['ONE PIECE 巻12', 200002],
['ONE PIECE 巻13', 200005],
['ONE PIECE 巻14', 200007],
['ONE PIECE 巻15', 200009],
['ONE PIECE 巻16', 200012],
['ONE PIECE 巻17', 200102],
['ONE PIECE 巻18', 200104],
['ONE PIECE 巻19', 200107],
['ONE PIECE 巻20', 200109],
['ONE PIECE 巻21', 200112],
['ONE PIECE RED', 200201],
['ONE PIECE 巻22', 200202],
['ONE PIECE 巻23', 200204],
['ONE PIECE 巻24', 200207],
['ONE PIECE 巻25', 200209],
['ONE PIECE 巻26', 200212],
['ONE PIECE 巻27', 200302],
['ONE PIECE 巻28', 200305],
['ONE PIECE 巻29', 200307],
['ONE PIECE 巻30', 200310],
['ONE PIECE COLOR WALK 2', 200311],
['ONE PIECE 巻31', 200312],
['ONE PIECE 巻32', 200403],
['ONE PIECE 巻33', 200406],
['ONE PIECE 巻34', 200408],
['ONE PIECE 巻35', 200411],
['ONE PIECE 巻36', 200502],
['ONE PIECE THE FIRST LOG', 200504],
['ONE PIECE 巻37', 200505],
['ONE PIECE 巻38', 200507],
['ONE PIECE 巻39', 200511],
['ONE PIECE THE 2ND LOG “SANJI', 200511],
['ONE PIECE 巻40', 200512],
['ONE PIECE THE 3RD LOG “NAMI', 200512],
['ONE PIECE THE 4TH LOG “GRAND LINE', 200601],
['ONE PIECE COLOR WALK 3', 200601],
['ONE PIECE THE 5TH LOG “CHOPPER', 200602],
['ONE PIECE THE 6TH LOG AR“ABASTA', 200603],
['ONE PIECE 巻41', 200604],
['ONE PIECE THE 7TH LOG “VIVI', 200605],
['ONE PIECE 巻42', 200607],
['ONE PIECE 巻43', 200609],
['ONE PIECE THE MOVIE カラクリ城のメカ巨兵', 200611],
['ONE PIECE 巻44', 200612],
['ONE PIECE 巻45', 200703],
['ONE PIECE 巻46', 200707],

['ONE PIECE 巻47', 200709],
['ONE PIECE 巻48', 200712],
['ONE PIECE 巻49', 200803],
['劇場版 ONE PIECE', 200803],
['ONE PIECE 巻50', 200806],
['ONE PIECE THE 10TH LOG “BELL', 200806],
['ONE PIECE 巻51', 200809],
['ONE PIECE 巻52', 200812],
['ONE PIECE 巻53', 200903],
['ONE PIECE THE 11TH LOG “WATER SEVEN', 200904],
['ONE PIECE 巻54', 200906],
['ONE PIECE THE 12TH LOG “ROCKET MAN', 200906],
['ONE PIECE THE 13TH LOG “NICO ROBIN', 200907],
['ONE PIECE THE 14TH LOG “FRANKY', 200908],
['ONE PIECE 巻55', 200909],
['ONE PIECE THE MOVIE エピソードオブチョッパー+冬に咲く、奇跡の桜', 200911],
['ONE PIECE 巻56', 200912],
['ONE PIECE THE 5TH LOG “CHOPPER', 201001],
['ONE PIECE THE 4TH LOG “GRAND LINE', 201001],
['ONE PIECE THE 7TH LOG “VIVI', 201002],
['ONE PIECE THE 6TH LOG “ARABASTA', 201002],
['ONE PIECE 巻57', 201003],
['ONE PIECE THE 9TH LOG “GOD', 201003],
['ONE PIECE THE 8TH LOG “SKYPIEA', 201003],
['ONE PIECE COLOR WALK 4', 201003],
['ONE PIECE 巻58', 201006],
['ONE PIECE 巻59', 201008],
['ONE PIECE 巻60', 201011],
['ONE PIECE FILM STRONG WORLD 下', 201012],
['ONE PIECE FILM STRONG WORLD 下', 201012],
['ONE PIECE FILM STRONG WORLD 上', 201012],
['ONE PIECE COLOR WALK 5', 201012],
['ONE PIECE 巻61', 201102],
['ONE PIECE THE MOVIEデッドエンドの冒険', 201102],
['ONE PIECE呪われた聖剣', 201102],
['ONE PIECE THE 15TH LOG“THRILLER BARK”', 201102],
['ONE PIECE THE 16TH LOG“THRILLER BROOK”', 201102],
['ONE PIECE 巻62', 201105],
['ONE PIECE 巻63', 201108],
['ONE PIECE 巻64', 201111],
['ONE PIECE 巻65', 201202],
['ONE PIECE 巻66', 201205],
['ONE PIECE 巻67', 201208],
['ONE PIECE 総集編 THE11TH LOG', 201208],
['ONE PIECE 総集編 THE12TH LOG', 201208],
['ONE PIECE 総集編 THE13TH LOG', 201208],
['ONE PIECE 総集編 THE14TH LOG', 201208],
['ONE PIECE 総集編 THE15TH LOG', 201208],
['ONE PIECE 総集編 THE16TH LOG', 201208],
['ONE PIECE 総集編 THE17TH LOG', 201209],
['ONE PIECE 総集編 THE18TH LOG', 201210],
['ONE PIECE 巻68', 201211],
['ONE PIECE 総集編 THE19TH LOG', 201211],
['ONE PIECE EXTRA LOG 1', 201212],
['ONE PIECE 総集編 THE20TH LOG', 201212],
['ONE PIECE 巻69', 201303],
['ONE PIECE 巻70', 201306],
['ONE PIECE FILM Z 上', 201307],

['ONE PIECE FILM Z 下', 201307],
['ONE PIECE 巻71', 201308],
['ONE PIECE 巻72', 201311],
['ONE PIECE COLOR WALK 6', 201401],
['ONE PIECE 巻73', 201403],
['ONE PIECE 巻74', 201406],
['ONE PIECE 巻75', 201409],
['ファンが選ぶONE PIECE“涙”ベスト10!! 1', 201412],
['ファンが選ぶONE PIECE”涙”ベスト10!! II', 201412],
['ONE PIECE 巻76', 201412],
['ONE PIECE 巻77', 201504],
['ONE PIECE 巻78', 201507],
['ONE PIECE 巻79', 201510],
['ONE PIECE 巻80', 201512],
['ONE PIECE 巻81', 201604],
['ONE PIECE 巻82', 201607],
['ONE PIECE COLOR WALK 7', 201607],
['ONE PIECE 巻83', 201611],
['ONE PIECE 巻84', 201702],
['ONE PIECE 巻85', 201705],
['ONE PIECE THE 21ST LOG“2 YEARS LATER”', 201706],
['ONE PIECE FILM GOLD 上', 201707],
['ONE PIECE FILM GOLD 下', 201707],
['ONE PIECE 総集編 THE 22ND LOG', 201707],
['ONE PIECE 巻86', 201708],
['ONE PIECE 総集編 THE 23RD LOG', 201708],
['ONE PIECE 総集編 THE 24TH LOG', 201709],
['ONE PIECE総集編 THE 25TH LOG', 201710],
['ONE PIECE 巻87', 201711],
['ONE PIECE総集編 THE 26TH LOG', 201711],
['ONE PIECE総集編 THE 27TH LOG', 201712],
['ONE PIECE総集編 THE 28TH LOG', 201801],
['ONE PIECE 巻88', 201803],
['ONE PIECE 巻89', 201806],
['ワンピースパーティー = ONE PIECE PARTY 4', 201806],
['ONE PIECE 巻90', 201809],
['ONE PIECE 巻91', 201812],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻1', 201812],
['ONE PIECE 巻92', 201903],
['ワンピースパーティー = ONE PIECE PARTY 5', 201903],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻3', 201907],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻2', 201907],
['ONE PIECE 巻93', 201907],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻1", 201907],
['ONE PIECE 巻94', 201910],
['ONE PIECE 巻95', 201912],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻2", 202004],
['ワンピースパーティー = ONE PIECE PARTY 6', 202004],
['ONE PIECE 巻96', 202004],
['ONE PIECE学園 1', 202004],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 上', 202005],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 下', 202005],
['ONE PIECE 巻97', 202009],
['ONE PIECE 巻98', 202102],
['ONE PIECE 巻99', 202106],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻3", 202108],
['ONE PIECE 巻100', 202109],
['ONE PIECE学園 3', 202112],

```
['ONE PIECE 巻101', 202112],
['ONE PIECE 巻102', 202204],
['ONE PIECE学園 4', 202204],
['ONE PIECEエピソードエース 1', 202208],
['ONE PIECE 巻103', 202208],
['ONE PIECEエピソードエース 2', 202208],
['ONE PIECE学園 5', 202211],
['ONE PIECE 巻104', 202211],
['ONE PIECE学園 6', 202303],
['ONE PIECE 巻105', 202303]]
```

In [ ]:
```
onepiece
```

Out[ ]:
```
[['ONE PIECE FILM STRONG WORLD 下', 201012],
 ['ONE PIECE FILM STRONG WORLD 上', 201012],
 ['ONE PIECE THE MOVIE カラクリ城のメカ巨兵', 200611],
 ['ONE PIECE THE MOVIE エピソードオブチョッパー+冬に咲く、奇跡の桜', 200911],
 ['ONE PIECE 巻61', 201102],
 ['ONE PIECE 巻60', 201011],
 ['ONE PIECE 巻59', 201008],
 ['ONE PIECE 巻58', 201006],
 ['ONE PIECE 巻56', 200912],
 ['ONE PIECE 巻55', 200909],
 ['ONE PIECE 巻54', 200906],
 ['ONE PIECE 巻53', 200903],
 ['ONE PIECE 巻52', 200812],
 ['ONE PIECE 巻51', 200809],
 ['ONE PIECE 巻50', 200806],
 ['ONE PIECE 巻49', 200803],
 ['ONE PIECE 巻48', 200712],
 ['ONE PIECE 巻47', 200709],
 ['ONE PIECE 巻45', 200703],
 ['ONE PIECE 巻46', 200707],
 ['ONE PIECE 巻57', 201003],
 ['ONE PIECE 巻44', 200612],
 ['ONE PIECE 巻43', 200609],
 ['ONE PIECE 巻42', 200607],
 ['ONE PIECE 巻41', 200604],
 ['ONE PIECE 巻40', 200512],
 ['ONE PIECE 巻39', 200511],
 ['ONE PIECE 巻38', 200507],
 ['ONE PIECE 巻36', 200502],
 ['ONE PIECE 巻35', 200411],
 ['ONE PIECE 巻34', 200408],
 ['ONE PIECE 巻33', 200406],
 ['ONE PIECE 巻32', 200403],
 ['ONE PIECE 巻31', 200312],
 ['ONE PIECE 巻30', 200310],
 ['ONE PIECE 巻28', 200305],
 ['ONE PIECE 巻27', 200302],
 ['ONE PIECE 巻26', 200212],
 ['ONE PIECE 巻25', 200209],
 ['ONE PIECE 巻24', 200207],
 ['ONE PIECE 巻23', 200204],
 ['ONE PIECE 巻37', 200505],
 ['ONE PIECE THE 2ND LOG “SANJI', 200511],
 ['ONE PIECE THE 14TH LOG “FRANKY', 200908],
 ['ONE PIECE THE 13TH LOG “NICO ROBIN', 200907],
```

['ONE PIECE THE 12TH LOG “ROCKET MAN', 200906],
['ONE PIECE THE 11TH LOG “WATER SEVEN', 200904],
['ONE PIECE THE 9TH LOG “GOD', 201003],
['ONE PIECE THE 8TH LOG “SKYPIEA', 201003],
['ONE PIECE THE 7TH LOG “VIVI', 201002],
['ONE PIECE THE 7TH LOG “VIVI', 200605],
['ONE PIECE THE 6TH LOG “ARABASTA', 201002],
['ONE PIECE THE 6TH LOG AR“ABASTA', 200603],
['ONE PIECE THE 5TH LOG “CHOPPER', 201001],
['ONE PIECE THE 5TH LOG “CHOPPER', 200602],
['ONE PIECE THE 4TH LOG “GRAND LINE', 201001],
['ONE PIECE THE 4TH LOG “GRAND LINE', 200601],
['ONE PIECE THE 3RD LOG “NAMI', 200512],
['ONE PIECE THE FIRST LOG', 200504],
['劇場版 ONE PIECE', 200803],
['ONE PIECE THE 10TH LOG “BELL', 200806],
['ONE PIECE EXTRA LOG 1', 201212],
['ONE PIECE FILM Z 上', 201307],
['ONE PIECE FILM Z 下', 201307],
['ONE PIECE 巻63', 201108],
['ONE PIECE 巻64', 201111],
['ONE PIECE 巻65', 201202],
['ONE PIECE 巻66', 201205],
['ONE PIECE 巻67', 201208],
['ONE PIECE 巻68', 201211],
['ONE PIECE 巻69', 201303],
['ONE PIECE 巻70', 201306],
['ONE PIECE 巻71', 201308],
['ONE PIECE 総集編 THE11TH LOG', 201208],
['ONE PIECE 総集編 THE12TH LOG', 201208],
['ONE PIECE 総集編 THE13TH LOG', 201208],
['ONE PIECE 総集編 THE14TH LOG', 201208],
['ONE PIECE 総集編 THE15TH LOG', 201208],
['ONE PIECE 総集編 THE16TH LOG', 201208],
['ONE PIECE 総集編 THE17TH LOG', 201209],
['ONE PIECE 総集編 THE18TH LOG', 201210],
['ONE PIECE 総集編 THE19TH LOG', 201211],
['ONE PIECE 総集編 THE20TH LOG', 201212],
['ONE PIECE THE MOVIEデッドエンドの冒険', 201102],
['ONE PIECE呪われた聖剣', 201102],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻3', 201907],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻2', 201907],
['ONE PIECE 巻93', 201907],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻1", 201907],
['ONE PIECE 巻94', 201910],
['ONE PIECE 巻95', 201912],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 上', 202005],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 下', 202005],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻2", 202004],
['ワンピースパーティー = ONE PIECE PARTY 6', 202004],
['ONE PIECE 巻96', 202004],
['ONE PIECE学園 1', 202004],
['ONE PIECE 巻97', 202009],
['ONE PIECE 巻98', 202102],
['ONE PIECE 巻99', 202106],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻3", 202108],
['ONE PIECE 巻100', 202109],
['ONE PIECE学園 3', 202112],

['ONE PIECE 巻101', 202112],
['ONE PIECE 巻102', 202204],
['ONE PIECE学園 4', 202204],
['ONE PIECEエピソードエース 1', 202208],
['ONE PIECE 巻103', 202208],
['ONE PIECEエピソードエース 2', 202208],
['ONE PIECE学園 5', 202211],
['ONE PIECE 巻104', 202211],
['ONE PIECE学園 6', 202303],
['ONE PIECE 巻105', 202303],
['ONE PIECE 巻62', 201105],
['ONE PIECE THE 15TH LOG“THRILLER BARK”', 201102],
['ONE PIECE THE 16TH LOG“THRILLER BROOK”', 201102],
['ONE PIECE 巻72', 201311],
['ONE PIECE COLOR WALK 6', 201401],
['ONE PIECE 巻73', 201403],
['ONE PIECE 巻74', 201406],
['ONE PIECE 巻75', 201409],
['ファンが選ぶONE PIECE“涙”ベスト10!! 1', 201412],
['ファンが選ぶONE PIECE”涙”ベスト10!! II', 201412],
['ONE PIECE 巻77', 201504],
['ONE PIECE 巻78', 201507],
['ONE PIECE 巻76', 201412],
['ONE PIECE 巻79', 201510],
['ONE PIECE 巻80', 201512],
['ONE PIECE 巻81', 201604],
['ONE PIECE 巻82', 201607],
['ONE PIECE COLOR WALK 7', 201607],
['ONE PIECE 巻83', 201611],
['ONE PIECE 巻84', 201702],
['ONE PIECE 巻85', 201705],
['ONE PIECE THE 21ST LOG“2 YEARS LATER”', 201706],
['ONE PIECE FILM GOLD 上', 201707],
['ONE PIECE FILM GOLD 下', 201707],
['ONE PIECE 総集編 THE 22ND LOG', 201707],
['ONE PIECE 巻86', 201708],
['ONE PIECE 総集編 THE 23RD LOG', 201708],
['ONE PIECE 総集編 THE 24TH LOG', 201709],
['ONE PIECE総集編 THE 25TH LOG', 201710],
['ONE PIECE 巻87', 201711],
['ONE PIECE総集編 THE 26TH LOG', 201711],
['ONE PIECE総集編 THE 27TH LOG', 201712],
['ONE PIECE総集編 THE 28TH LOG', 201801],
['ONE PIECE 巻88', 201803],
['ONE PIECE 巻91', 201812],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻1', 201812],
['ONE PIECE 巻92', 201903],
['ワンピースパーティー = ONE PIECE PARTY 5', 201903],
['ONE PIECE 巻90', 201809],
['ONE PIECE 巻89', 201806],
['ワンピースパーティー = ONE PIECE PARTY 4', 201806],
['ONE PIECE COLOR WALK 5', 201012],
['ONE PIECE COLOR WALK 4', 201003],
['ONE PIECE COLOR WALK 3', 200601],
['ONE PIECE 巻22', 200202],
['ONE PIECE 巻21', 200112],
['ONE PIECE 巻20', 200109],
['ONE PIECE 巻19', 200107],

```
['ONE PIECE 巻18', 200104],
['ONE PIECE 巻17', 200102],
['ONE PIECE 巻16', 200012],
['ONE PIECE 巻6', 199812],
['ONE PIECE 巻4', 199808],
['ONE PIECE 巻5', 199810],
['ONE PIECE 巻3', 199806],
['ONE PIECE 巻2', 199804],
['ONE PIECE 巻1', 199712],
['ONE PIECE COLOR WALK 2', 200311],
['ONE PIECE RED', 200201],
['ONE PIECE 巻29', 200307],
['ONE PIECE 巻15', 200009],
['ONE PIECE 巻14', 200007],
['ONE PIECE 巻13', 200005],
['ONE PIECE 巻12', 200002],
['ONE PIECE 巻11', 199912],
['ONE PIECE 巻10', 199910],
['ONE PIECE 巻9', 199907],
['ONE PIECE 巻8', 199905],
['ONE PIECE 巻7', 199903]]
```

マージソート

In [ ]:

```python
def merge_sort(arr):
  #リストが一つの場合そのまま返す
    if len(arr) <= 1:
        return arr
    # 中央で分割
    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    # 再帰的に分割してソート
    left_half = merge_sort(left_half)
    right_half = merge_sort(right_half)

    # 結合
    return merge(left_half, right_half)

def merge(left, right):
    result = []
    left_idx, right_idx = 0, 0

    while left_idx < len(left) and right_idx < len(right):
        if left[left_idx][1] < right[right_idx][1]:
            result.append(left[left_idx])
            left_idx += 1
        else:
            result.append(right[right_idx])
            right_idx += 1

    result.extend(left[left_idx:])
    result.extend(right[right_idx:])

    return result
```

In [ ]:

```python
# 時間計測開始
start = time.time()

# 新しいリストを作り,そこにソートしたものが入っていく
```

```
sorted_onepiece = merge_sort(onepiece)

# 時間計測終了
print("計算時間: ", time.time() - start)

sorted_onepiece
```

```
計算時間:  0.0006506443023681641
```

Out[ ]:
```
[['ONE PIECE 巻1', 199712],
 ['ONE PIECE 巻2', 199804],
 ['ONE PIECE 巻3', 199806],
 ['ONE PIECE 巻4', 199808],
 ['ONE PIECE 巻5', 199810],
 ['ONE PIECE 巻6', 199812],
 ['ONE PIECE 巻7', 199903],
 ['ONE PIECE 巻8', 199905],
 ['ONE PIECE 巻9', 199907],
 ['ONE PIECE 巻10', 199910],
 ['ONE PIECE 巻11', 199912],
 ['ONE PIECE 巻12', 200002],
 ['ONE PIECE 巻13', 200005],
 ['ONE PIECE 巻14', 200007],
 ['ONE PIECE 巻15', 200009],
 ['ONE PIECE 巻16', 200012],
 ['ONE PIECE 巻17', 200102],
 ['ONE PIECE 巻18', 200104],
 ['ONE PIECE 巻19', 200107],
 ['ONE PIECE 巻20', 200109],
 ['ONE PIECE 巻21', 200112],
 ['ONE PIECE RED', 200201],
 ['ONE PIECE 巻22', 200202],
 ['ONE PIECE 巻23', 200204],
 ['ONE PIECE 巻24', 200207],
 ['ONE PIECE 巻25', 200209],
 ['ONE PIECE 巻26', 200212],
 ['ONE PIECE 巻27', 200302],
 ['ONE PIECE 巻28', 200305],
 ['ONE PIECE 巻29', 200307],
 ['ONE PIECE 巻30', 200310],
 ['ONE PIECE COLOR WALK 2', 200311],
 ['ONE PIECE 巻31', 200312],
 ['ONE PIECE 巻32', 200403],
 ['ONE PIECE 巻33', 200406],
 ['ONE PIECE 巻34', 200408],
 ['ONE PIECE 巻35', 200411],
 ['ONE PIECE 巻36', 200502],
 ['ONE PIECE THE FIRST LOG', 200504],
 ['ONE PIECE 巻37', 200505],
 ['ONE PIECE 巻38', 200507],
 ['ONE PIECE THE 2ND LOG “SANJI', 200511],
 ['ONE PIECE 巻39', 200511],
 ['ONE PIECE THE 3RD LOG “NAMI', 200512],
 ['ONE PIECE 巻40', 200512],
 ['ONE PIECE COLOR WALK 3', 200601],
 ['ONE PIECE THE 4TH LOG “GRAND LINE', 200601],
 ['ONE PIECE THE 5TH LOG “CHOPPER', 200602],
 ['ONE PIECE THE 6TH LOG AR“ABASTA', 200603],
 ['ONE PIECE 巻41', 200604],
```

['ONE PIECE THE 7TH LOG “VIVI', 200605],
['ONE PIECE 巻42', 200607],
['ONE PIECE 巻43', 200609],
['ONE PIECE THE MOVIE カラクリ城のメカ巨兵', 200611],
['ONE PIECE 巻44', 200612],
['ONE PIECE 巻45', 200703],
['ONE PIECE 巻46', 200707],
['ONE PIECE 巻47', 200709],
['ONE PIECE 巻48', 200712],
['劇場版 ONE PIECE', 200803],
['ONE PIECE 巻49', 200803],
['ONE PIECE THE 10TH LOG “BELL', 200806],
['ONE PIECE 巻50', 200806],
['ONE PIECE 巻51', 200809],
['ONE PIECE 巻52', 200812],
['ONE PIECE 巻53', 200903],
['ONE PIECE THE 11TH LOG “WATER SEVEN', 200904],
['ONE PIECE THE 12TH LOG “ROCKET MAN', 200906],
['ONE PIECE 巻54', 200906],
['ONE PIECE THE 13TH LOG “NICO ROBIN', 200907],
['ONE PIECE THE 14TH LOG “FRANKY', 200908],
['ONE PIECE 巻55', 200909],
['ONE PIECE THE MOVIE エピソードオブチョッパー+冬に咲く、奇跡の桜', 200911],
['ONE PIECE 巻56', 200912],
['ONE PIECE THE 4TH LOG “GRAND LINE', 201001],
['ONE PIECE THE 5TH LOG “CHOPPER', 201001],
['ONE PIECE THE 6TH LOG “ARABASTA', 201002],
['ONE PIECE THE 7TH LOG “VIVI', 201002],
['ONE PIECE COLOR WALK 4', 201003],
['ONE PIECE THE 8TH LOG “SKYPIEA', 201003],
['ONE PIECE THE 9TH LOG “GOD', 201003],
['ONE PIECE 巻57', 201003],
['ONE PIECE 巻58', 201006],
['ONE PIECE 巻59', 201008],
['ONE PIECE 巻60', 201011],
['ONE PIECE COLOR WALK 5', 201012],
['ONE PIECE FILM STRONG WORLD 上', 201012],
['ONE PIECE FILM STRONG WORLD 下', 201012],
['ONE PIECE THE 16TH LOG“THRILLER BROOK”', 201102],
['ONE PIECE THE 15TH LOG“THRILLER BARK”', 201102],
['ONE PIECE呪われた聖剣', 201102],
['ONE PIECE THE MOVIEデッドエンドの冒険', 201102],
['ONE PIECE 巻61', 201102],
['ONE PIECE 巻62', 201105],
['ONE PIECE 巻63', 201108],
['ONE PIECE 巻64', 201111],
['ONE PIECE 巻65', 201202],
['ONE PIECE 巻66', 201205],
['ONE PIECE 総集編 THE16TH LOG', 201208],
['ONE PIECE 総集編 THE15TH LOG', 201208],
['ONE PIECE 総集編 THE14TH LOG', 201208],
['ONE PIECE 総集編 THE13TH LOG', 201208],
['ONE PIECE 総集編 THE12TH LOG', 201208],
['ONE PIECE 総集編 THE11TH LOG', 201208],
['ONE PIECE 巻67', 201208],
['ONE PIECE 総集編 THE17TH LOG', 201209],
['ONE PIECE 総集編 THE18TH LOG', 201210],
['ONE PIECE 総集編 THE19TH LOG', 201211],

['ONE PIECE 巻68', 201211],
['ONE PIECE 総集編 THE20TH LOG', 201212],
['ONE PIECE EXTRA LOG 1', 201212],
['ONE PIECE 巻69', 201303],
['ONE PIECE 巻70', 201306],
['ONE PIECE FILM Z 下', 201307],
['ONE PIECE FILM Z 上', 201307],
['ONE PIECE 巻71', 201308],
['ONE PIECE 巻72', 201311],
['ONE PIECE COLOR WALK 6', 201401],
['ONE PIECE 巻73', 201403],
['ONE PIECE 巻74', 201406],
['ONE PIECE 巻75', 201409],
['ONE PIECE 巻76', 201412],
['ファンが選ぶONE PIECE”涙”ベスト10!! II', 201412],
['ファンが選ぶONE PIECE“涙”ベスト10!! 1', 201412],
['ONE PIECE 巻77', 201504],
['ONE PIECE 巻78', 201507],
['ONE PIECE 巻79', 201510],
['ONE PIECE 巻80', 201512],
['ONE PIECE 巻81', 201604],
['ONE PIECE COLOR WALK 7', 201607],
['ONE PIECE 巻82', 201607],
['ONE PIECE 巻83', 201611],
['ONE PIECE 巻84', 201702],
['ONE PIECE 巻85', 201705],
['ONE PIECE THE 21ST LOG“2 YEARS LATER”', 201706],
['ONE PIECE 総集編 THE 22ND LOG', 201707],
['ONE PIECE FILM GOLD 下', 201707],
['ONE PIECE FILM GOLD 上', 201707],
['ONE PIECE 総集編 THE 23RD LOG', 201708],
['ONE PIECE 巻86', 201708],
['ONE PIECE 総集編 THE 24TH LOG', 201709],
['ONE PIECE総集編 THE 25TH LOG', 201710],
['ONE PIECE総集編 THE 26TH LOG', 201711],
['ONE PIECE 巻87', 201711],
['ONE PIECE総集編 THE 27TH LOG', 201712],
['ONE PIECE総集編 THE 28TH LOG', 201801],
['ONE PIECE 巻88', 201803],
['ワンピースパーティー = ONE PIECE PARTY 4', 201806],
['ONE PIECE 巻89', 201806],
['ONE PIECE 巻90', 201809],
['ONE PIECEコビー似の小日山 : ウリふたつなぎの大秘宝 巻1', 201812],
['ONE PIECE 巻91', 201812],
['ワンピースパーティー = ONE PIECE PARTY 5', 201903],
['ONE PIECE 巻92', 201903],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻1", 201907],
['ONE PIECE 巻93', 201907],
['ONE PIECEコビー似の小日山 : ウリふたつなぎの大秘宝 巻2', 201907],
['ONE PIECEコビー似の小日山 : ウリふたつなぎの大秘宝 巻3', 201907],
['ONE PIECE 巻94', 201910],
['ONE PIECE 巻95', 201912],
['ONE PIECE学園 1', 202004],
['ONE PIECE 巻96', 202004],
['ワンピースパーティー = ONE PIECE PARTY 6', 202004],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻2", 202004],
['劇場版ONE PIECE STAMPEDE : アニメコミックス 下', 202005],
['劇場版ONE PIECE STAMPEDE : アニメコミックス 上', 202005],

['ONE PIECE 巻97', 202009],
['ONE PIECE 巻98', 202102],
['ONE PIECE 巻99', 202106],
["Fischer's×ONE PIECE 7つなぎの大秘宝 巻3", 202108],
['ONE PIECE 巻100', 202109],
['ONE PIECE 巻101', 202112],
['ONE PIECE学園 3', 202112],
['ONE PIECE学園 4', 202204],
['ONE PIECE 巻102', 202204],
['ONE PIECEエピソードエース 2', 202208],
['ONE PIECE 巻103', 202208],
['ONE PIECEエピソードエース 1', 202208],
['ONE PIECE 巻104', 202211],
['ONE PIECE学園 5', 202211],
['ONE PIECE 巻105', 202303],
['ONE PIECE学園 6', 202303]]

In [ ]:

1-7. アルゴリズム - 探索アルゴリズム（リスト探索）

あるデータセットから所望のデータを探すとき，どのようなアルゴリズムを使うかによって効率的に探索できるのかどうかが変わってくる．実際にコードを動かして確認してみよう．

データセットの準備

以下では，メディア芸術データベースのデータを利用し，「ONE PIECE」のタイトルを出版日から探索していく． 上記のリンクから「マンガ単行本」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

※「1-4. 単回帰分析」，「1-4. ロジスティック回帰分析」，「1-5. 1〜3次元の図表化」，「1-5. 関係性の可視化（ネットワーク構造）」，「1-7. ソートアルゴリズム」，「1-7. 探索アルゴリズム」，「2-5. データ加工」，「3-3. 機械学習」は同じデータセットを利用するため，もし同じものを持っている場合は以下の取得作業は不要である．そちらをアップロードしよう．

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう．（およそ30分程度）

In [ ]:
```
import json
import copy
import time
```

In [ ]:
```
# ファイルを開き,読み込む
j_file=open('/content/metadata_cm-item_cm101_00001.json')
data = json.load(j_file)

# 漫画のデータは'@graph'配下にあるので,そこを切りだす
comics=data['@graph']
```

In [ ]:
```
# ONE PIECEのタイトルを格納するリスト
onepiece=[]

# comicsデータ内の各要素（comic）を順番に見ていく
# 「タイトルに"ONE PIECE"が入っている」かつ「出版日のデータがあるもの」を"onepiece"リストに格納していく
for comic in comics:
  if 'label' in comic.keys() and 'ONE PIECE' in comic['label']:
    if('datePublished' in comic.keys()):

      date=comic['datePublished']
      date=date.replace('-','')
      date=date[0:6]
      onepiece.append([comic['label'],int(date)])
```

のちに紹介する二分探索では，探索するデータがソートされている必要がある． そのため，ソート済みのデータも用意しておく．

In [ ]:
```
# onepieceリストのコピーを作り,それをソートする
sorted_onepiece = copy.deepcopy(onepiece)

# 時間計測開始
start = time.time()
```

```
# 要素を頭から順番に見ていく
for i in range(len(sorted_onepiece)-1):
  # sorted_onepiece[0]~[終端]の中で,値の大きいものを次の値と交換するということを繰り返し,最大値を持
  # 終端側(値の大きい方)のi個がソート済みになるので,新しく最大値が終端に追いやられるたびに終端の位置
  for j in range(len(sorted_onepiece)-1-i):
    if sorted_onepiece[j][1]>sorted_onepiece[j+1][1]:
      tmp1=sorted_onepiece[j]
      tmp2=sorted_onepiece[j+1]
      sorted_onepiece[j]=tmp2
      sorted_onepiece[j+1]=tmp1

# 時間計測終了
print("計算時間: ", time.time() - start)

sorted_onepiece
```

```
計算時間:  0.0008900165557861328
```

Out[ ]:
```
[['ONE PIECE冒険ロマン解析書', 199911],
 ['ONE PIECEキャラクター心理分析書', 200111],
 ['ONE PIECE RED', 200201],
 ['ONE PIECE 巻23', 200204],
 ['ONE PIECE 巻25', 200209],
 ['ONE PIECE 巻26', 200212],
 ['ONE PIECE 巻27', 200302],
 ['ONE PIECE熱血キャラ解析書', 200307],
 ['ONE PIECE 巻35', 200411],
 ['ONE PIECE THE 3RD LOG “NAMI', 200512],
 ['ONE PIECE COLOR WALK 3', 200601],
 ['ONE PIECE THE 4TH LOG “GRAND LINE', 200601],
 ['劇場版 ONE PIECE', 200803],
 ['ONE PIECE THE 13TH LOG “NICO ROBIN', 200907],
 ['ONE PIECE 巻55', 200909],
 ['ONE PIECE THE 6TH LOG “ARABASTA', 201002],
 ['ONE PIECE COLOR WALK 4', 201003],
 ['ONE PIECE THE 8TH LOG “SKYPIEA', 201003],
 ['ONE PIECE FILM STRONG WORLD 上', 201012],
 ['ONE PIECE BLUE DEEP CHARACTERS WORLD', 201203],
 ['ONE PIECE 総集編 THE11TH LOG', 201208],
 ['ONE PIECE 総集編 THE13TH LOG', 201208],
 ['ONE PIECE EXTRA LOG 1', 201212],
 ['ONE PIECE 総集編 THE20TH LOG', 201212],
 ['ONE PIECE COLOR WALK 6', 201401],
 ['ONE PIECE最強でサイコーの名言集', 201403],
 ['ONE PIECE 巻74', 201406],
 ['ONE PIECE 500 QUIZ BOOK 2', 201409],
 ['ONE PIECE 巻78', 201507],
 ['ONE PIECE 巻82', 201607],
 ['ONE PIECE 巻83', 201611],
 ['ONE PIECE総集編 THE 25TH LOG', 201701],
 ['ONE PIECE THE 21ST LOG“2 YEARS LATER”', 201706],
 ['ONE PIECE 巻86', 201708],
 ['ONE PIECE magazine Vol.3', 201709],
 ['ONE PIECE 総集編 THE 24TH LOG', 201709],
 ['ONE PIECE総集編 THE 26TH LOG', 201711],
 ['ONE PIECE総集編 THE 27TH LOG', 201712],
 ['ONE PIECE総集編 THE 28TH LOG', 201801],
 ['ONE PIECE 巻89', 201806],
 ['ワンピースパーティー = ONE PIECE PARTY 4', 201806],
```

```
['ONE PIECE DOORS! 1', 201806],
['ONE PIECE DOORS! 2', 201807],
['ONE PIECE 巻91', 201812],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻1', 201812],
['ワンピースパーティー = ONE PIECE PARTY 5', 201903],
['ONE PIECEコビー似の小日山：ウリふたつなぎの大秘宝 巻2', 201907],
['ONE PIECE 巻93', 201907],
['ONE PIECE 巻94', 201910],
['ONE PIECE 巻95', 201912],
['ONE PIECE 巻96', 202004],
['ONE PIECE学園 1', 202004],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 上', 202005],
['劇場版ONE PIECE STAMPEDE：アニメコミックス 下', 202005],
['ONE PIECE 巻98', 202102]]
```

リスト探索

今回は2009年9月に出版された本について調べる.

In [ ]:
```
want_num=200909
```

線形探索

線形探索では，先頭（もしくは最後）から順番に要素を見ていって，条件に合致するものが見つかるまで続ける.

ソートしていない配列に対する線形探索

In [ ]:
```
# 時間計測開始
start = time.time()

# 所望のデータが見つかったかどうかを判定するフラグ
not_found=True

# 先頭から要素を見ていく
for i in range(len(onepiece)):
  # もし所望のデータが見つかったらprintし,フラグの値を変えてループから抜け出す
  if(onepiece[i][1]==want_num):
    print(onepiece[i][0])
    not_found=False
    break

if not_found:
  print('Not Found')

# 時間計測終了
print("計算時間: ", time.time() - start)
```

```
ONE PIECE 巻55
計算時間:  0.0015740394592285156
```

ソート済みの配列に対する線形探索

In [ ]:
```
# 時間計測開始
start = time.time()

# 所望のデータが見つかったかどうかを判定するフラグ
not_found=True

# 先頭から要素を見ていく
```

```
for i in range(len(sorted_onepiece)):
  # もし所望のデータが見つかったらprintし,フラグの値を変えてループから抜け出す
  if(sorted_onepiece[i][1]==want_num):
    print(sorted_onepiece[i][0])
    not_found=False
    break

if not_found:
  print('Not Found')

# 時間計測終了
print("計算時間: ", time.time() - start)
```

```
ONE PIECE 巻55
計算時間:  0.00213623046875
```

二分探索

二分探索では，ソート済みのデータに対して探索を行う．

In [ ]:

```
# 時間計測開始
start = time.time()

# 所望のデータが見つかったかどうかのフラグ
not_found=True

compare=sorted_onepiece

# リストを左半分・右半分に分割
mid=len(compare)//2
left_list=compare[:mid]
right_list=compare[mid+1:]

# リストの中央値と所望のデータを比べ,左右どちらのリストに所望のデータがあるのか判断
# 選んだ方のリストの中央値を見つけてまた半分にする…という作業を繰り返す
# 中央値が所望のデータになった(＝見つかった)時点で終了(成功).
# 見つからず,かつリストが半分にできなくなってしまった場合も終了(失敗).
while  len(compare)>0  and not_found:
  if compare[mid][1]>want_num:
    compare=left_list
    mid=len(compare)//2
    left_list=compare[:mid]
    right_list=compare[mid+1:]
  elif compare[mid][1]<want_num:
    compare=right_list
    mid=len(compare)//2
    left_list=compare[:mid]
    right_list=compare[mid+1:]
  else :
    not_found=False
    print(compare[mid][0])
    break

if not_found:
  print('Not Found')

# 時間計測終了
print("計算時間: ", time.time() - start)
```

```
ONE PIECE 巻55
計算時間:  0.0005247592926025391
```

2-5. データ加工 - データ型変換処理

以下では，メディア芸術データベースのデータを利用し，「ONE PIECE」のマンガタイトルを取得してPython上で扱える形式にしていく．上記のリンクから「マンガ単行本」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

※「1-4. 単回帰分析」，「1-4. ロジスティック回帰分析」，「1-5. 1〜3次元の図表化」，「1-5. 関係性の可視化（ネットワーク構造）」，「1-7. ソートアルゴリズム」，「1-7. 探索アルゴリズム」，「2-5. データ加工」，「3-3. 機械学習」は同じデータセットを利用するため，もし同じものを持っている場合は以下の取得作業は不要である．そちらをアップロードしよう．

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう．（およそ30分程度）

In [ ]:
```
import json
```

データの型変換処理

ここでは「ONE PIECE」のデータを抽出し，それぞれの名前と出版年月を抽出している．この際，出版年月を文字列（str型）から整数（int型）に型変換している．

In [ ]:
```
# ファイルを開き,読み込む
j_file=open('/content/metadata_cm-item_cm101_00001.json')
df = json.load(j_file)

# マンガのデータは'@graph'配下にあるので,そこを切りだす
comics=df['@graph']
```

In [ ]:
```
# ONE PIECEのタイトルを格納するリスト
onepiece=[]

# comicsデータ内の各要素(comic)を順番に見ていく
#「タイトルに"ONE PIECE"が入っている」かつ「出版日のデータがあるもの」を"onepiece"リストに格納していく
for comic in comics:
  if ('label' in comic.keys()) and ('ONE PIECE' in comic['label']):
    if('datePublished' in comic.keys()):

      date=comic['datePublished']
      date=date.replace('-','')
      # この時点ではまだdateは文字列
      date=date[0:6]
      # ここで新しいリストに入れ直すとき,int()で型変換(キャスト)を行なっている
      onepiece.append([comic['label'],int(date)])
```

この整形したデータを用いて1−7で解説をしている。

3-3. 機械学習の基礎と展望 - 機械学習、教師あり学習（学習データ、検証データ、ホールドアウト法、交差検証法）、教師なし学習、過学習

データセットの準備

以下では，メディア芸術データベースのデータを利用し，4つの出版社のマンガの名前から出版社を予測する教師あり学習や，ジャンプコミックスのページ数と価格からジャンプコミックスをクラスタリングする教師なし学習を行う． 上記のリンクから「マンガ単行本」および「マンガ雑誌単号」の「.json」ファイルをダウンロードした後，このノートブックにアップロードしよう．

※「マンガ単行本」については，「1-4. 単回帰分析」，「1-4. ロジスティック回帰分析」，「1-5. 1〜3次元の図表化」,「1-5. 関係性の可視化（ネットワーク構造）」，「1-7. ソートアルゴリズム」，「1-7. 探索アルゴリズム」，「2-5. データ加工」と同じデータセットを利用するため，すでにダウンロード済みの場合はそちらをアップロードしよう．

ファイルサイズがとても大きいためアップロードには時間がかかる．ファイル名が反映されたことを確認するだけでなく，ファイルアップロード時の画面の下部にあるアップロードの進捗を示す円形のバーが全て進行するまで待ってから作業しよう． （およそ30分程度）

ライブラリのインストールとデータの加工

In [ ]:
```
!pip install mecab-python3
!pip install unidic-lite

import re
import MeCab
import pandas as pd
import json
import random
from sklearn import svm
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import collections
from scipy.sparse import csr_matrix
```

```
Requirement already satisfied: mecab-python3 in /usr/local/lib/python3.10/dist-packages (1.0.8)
Requirement already satisfied: unidic-lite in /usr/local/lib/python3.10/dist-packages (1.0.8)
```

教師あり学習に用いるデータの取得

In [ ]:
```
j_file=open('/content/metadata_cm-item_cm101_00001.json')
df = json.load(j_file)
comics_supervised=df['@graph']
```

```
# マンガタイトルを格納
names=[]

# マンガの出版社を格納
publishers=[]

for comic in comics_supervised:
 #  データの加工
 if('publisher' in comic.keys() and 'label' in comic.keys() and ('集英社' in comic['publisher'] or'講
  names.append(comic['label'])
  publishers.append(comic['publisher'])
```

教師なし学習に用いるデータの取得

In [ ]:
```
j_file=open('/content/metadata_cm-item_cm102_00001.json')
df = json.load(j_file)
comics_nonsupervised = df['@graph']

# 雑誌単号の名前,厚さ,ページ数のリストを格納するリスト
jumps=[]

for comic in comics_nonsupervised:
 if('label' in comic.keys() and 'ジャンプ' in comic['label']):
  if('price' in comic.keys() and 'numberOfPages' in comic.keys()):
     price=re.sub(r"\D", "", comic['price'])
     page=re.sub(r"\D", "", comic['numberOfPages'].split('p')[0])
     jumps.append([comic['label'],int(price),int(page)])
```

タイトルから出版社を予測する教師あり学習

4つの出版社のマンガの名前から出版社を予測する教師あり学習を行ってみる。

In [ ]:
```
# 分かち書きを行うインスタンスwakatiの作成
wakati = MeCab.Tagger('-Owakati')

# 分かち書きしたタイトルを格納するリスト
wakati_names = []

for name in names:
 tmp = wakati.parse(name).strip()
 tmp = " ".join([token for token in tmp.split(" ") if token!="" and token.isnumeric()==False]) # 当
 wakati_names.append(tmp)
```

In [ ]:
```
# 出版社ごとに作品を分ける
syueisya = []
kodansya = []
kadokawa = []
syogakukan = []

X_str=[] #特徴量
y=[] #ラベル

for name, publisher in zip(wakati_names, publishers):
 if('集英社' in publisher):
  syueisya.append(name)
 elif('講談社' in publisher):
  kodansya.append(name)
 elif('KADOKAWA' in publisher):
  kadokawa.append(name)
 elif('小学館' in publisher):
  syogakukan.append(name)
```

```
# 4つの会社の作品の中から3000個ずつ選んで今回使うデータセットとする
import random
X_str.extend(random.sample(syueisya, 3000))
y.extend([0 for i in range(3000)])

X_str.extend(random.sample(kodansya, 3000))
y.extend([1 for i in range(3000)])

X_str.extend(random.sample(kadokawa, 3000))
y.extend([2 for i in range(3000)])

X_str.extend(random.sample(syogakukan, 3000))
y.extend([3 for i in range(3000)])

vec_tfidf = TfidfVectorizer()
X=vec_tfidf.fit_transform(X_str) #文字列を特徴量化

# テストデータと学習データに分ける
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

学習データ、検証データ

ホールドアウト法

In [ ]:
```
model=svm.SVC()
model.fit(X_train,y_train)
pred=model.predict(X_test)
print('正答率:',accuracy_score(y_test,pred))

cm=confusion_matrix(y_test,pred)
print(cm)
```

```
正答率: 0.6256666666666667
[[456 158  69  97]
 [ 81 505  90  99]
 [ 52  94 477  70]
 [ 70 170  73 439]]
```

交差検証法

データ全体をn分割して，そのうちの一つをテストデータ・残りのn-1個を学習データとしてテストデータのAccuracyを見る手法．　以下のコードは実行に5分ほどかかるため注意．

In [ ]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
cv = KFold(n_splits=5, random_state=0, shuffle=True)#5個に分割
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
print(scores)
```

```
[0.61125   0.60875   0.615     0.62333333 0.61458333]
```

平均値は次のようになる．

In [ ]:
```
np.mean(scores)
```

Out[ ]:
```
0.6145833333333333
```

厚さによるコミックス分類の教師なし学習

ジャンプコミックスのページ数と厚さからジャンプコミックスをクラスタリングする教師なし学習を行う。

In [ ]:
```
# jumpsの要素は[ジャンプコミックスの名前,価格,ページ数]というリスト

jdata=[]
for i in range (len(jumps)):
  jdata.append([jumps[i][2],jumps[i][1]])
```

In [ ]:
```
# 各要素ごとにどのジャンルのマンガなのか(=どのクラスタにあるべきなのか)という正解ラベルを作っていく
y_true = []

# マンガジャンルとインデックス(数字)を紐づけるためのリスト
cluster = []
for jump in jumps:
  tmp=jump[0].split(' ')[0]
  flag=1
  for i in range(len(cluster)):
    if cluster[i]==tmp:
      flag=0
      y_true.append(i)
      break
  if flag==1:
    cluster.append(tmp)
    y_true.append(len(cluster))

# クラスターがいくつあるのか確認する
# 本来はクラスタリングの際には正解のクラスタ数はわからないことが多いため,適切なクラスタ数を考えながら

collections.Counter(y_true)
```

Out[ ]:
```
Counter({1: 38,
         2: 11,
         3: 1465,
         4: 52,
         5: 17,
         6: 1,
         7: 26,
         8: 1,
         9: 5,
         10: 4,
         11: 2,
         12: 88,
         13: 34,
         14: 3,
         15: 1,
         16: 22,
         17: 220,
         18: 5,
         19: 1,
         20: 1,
         21: 797,
         22: 8,
         23: 2,
         24: 16,
         25: 27,
         26: 66,
         27: 8,
         28: 1,
         29: 3,
         30: 2,
         31: 1,
```

```
 32: 1,
 33: 9,
 34: 87,
 35: 20,
 36: 72,
 37: 173,
 38: 40,
 39: 1,
 40: 5,
 41: 16,
 42: 21,
 43: 15,
 0: 8,
 44: 17,
 45: 1,
 46: 43,
 47: 112,
 48: 1,
 49: 1,
 50: 1,
 51: 4,
 52: 1,
 53: 4,
 54: 1,
 55: 5,
 56: 1})
```

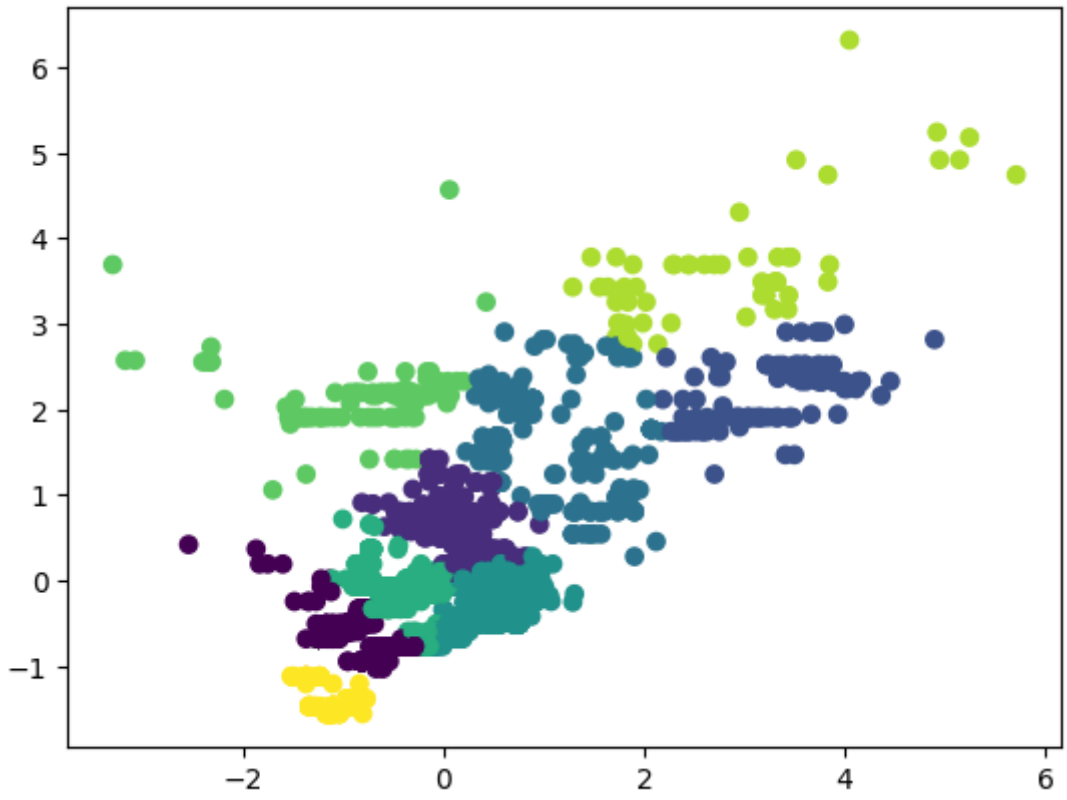
In [ ]:
```
# クラスタ数を決める
num_c=9

# 正規化
jdata_ss=StandardScaler().fit_transform(jdata)


model = KMeans(n_clusters=num_c)#K-平均法

# 重心計算
kmeans=model.fit(jdata_ss)

# 各データの所属クラスタを予測したもの
y_pred=kmeans.labels_


#プロット
x1 = [d[0] for d in jdata_ss]
x2 = [d[1] for d in jdata_ss]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

予測したクラスタによる分布図が以下.

In [ ]:
```
#横軸がページ,縦軸が厚さになっている
plt.scatter(x1,x2,c=y_pred)
```

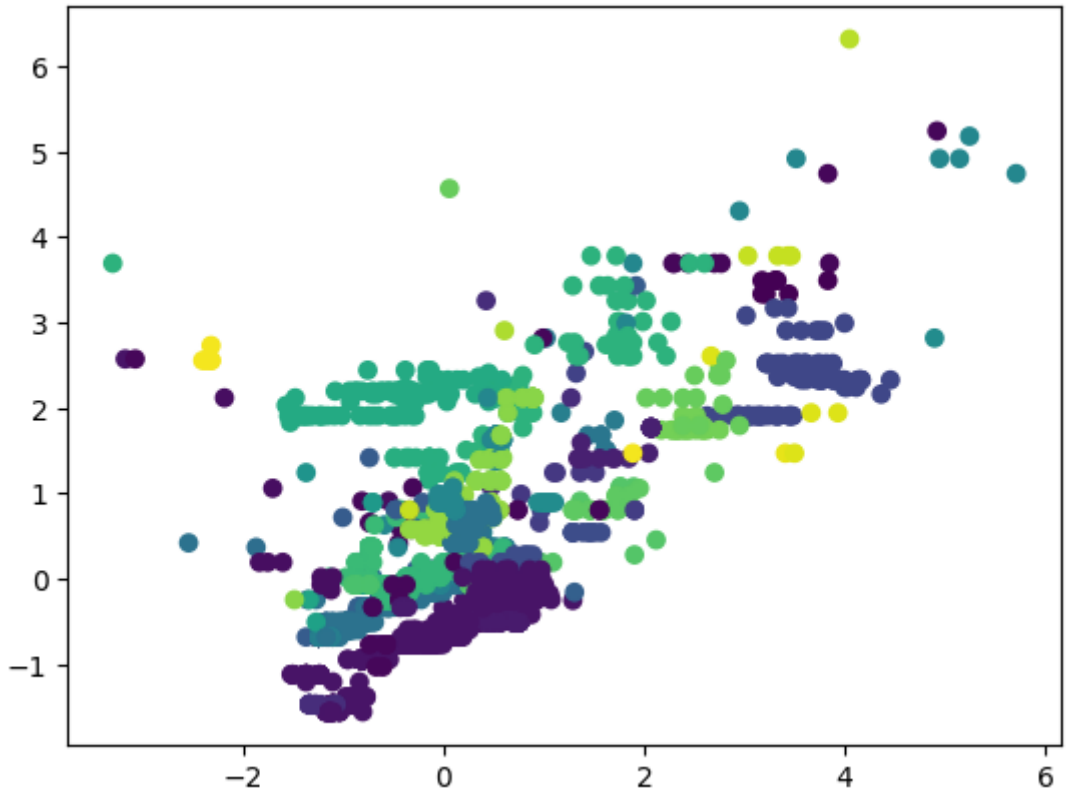
Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7cbef5548970>
```



正解のクラスタの分布図が以下.

In [ ]:
```
plt.scatter(x1, x2, c=y_true)
```

Out[ ]: <matplotlib.collections.PathCollection at 0x7cbe930cb280>



過学習

先ほどの教師あり学習のデータを改変して意図的に過学習を起こしてみる。 以下では，学習データが偏ってしまっているために汎化性能が上がらない例を見ていく.

In [ ]:
```
# 出版社ごとに作品を分ける
syueisya = []
kodansya = []
kadokawa = []
syogakukan = []

X_str=[] #特徴量
y=[] #ラベル

for name, publisher in zip(wakati_names, publishers):
  if('集英社' in publisher):
    syueisya.append(name)
  elif('講談社' in publisher):
    kodansya.append(name)
  elif('KADOKAWA' in publisher):
    kadokawa.append(name)
  elif('小学館' in publisher):
    syogakukan.append(name)

# 集英社の作品のみ5000個,残り3社の作品は1000個ずつ選んで今回使う学習データセットとする
# テストデータセットは4社とも100個ずつバランスよく選ぶ

import random
syueisya = random.sample(syueisya, 5100)
X_str.extend(syueisya)

kodansya = random.sample(kodansya, 1100)
X_str.extend(kodansya)

kadokawa = random.sample(kadokawa, 1100)
X_str.extend(kadokawa)

syogakukan = random.sample(syogakukan, 1100)
X_str.extend(syogakukan)

vec_tfidf = TfidfVectorizer()
X=vec_tfidf.fit(X_str) #文字列を特徴量化

# テストデータと学習データに分ける
X_train = []
X_test = []
y_train = []
y_test = []

for i, publisher in enumerate([syueisya, kodansya, kadokawa, syogakukan]):
    X_train_sub, X_test_sub, y_train_sub, y_test_sub = train_test_split(publisher, [i for j in rang
    print(type(vec_tfidf.transform(X_train_sub)[0]))
    X_train.extend(vec_tfidf.transform(X_train_sub).toarray())
    X_test.extend(vec_tfidf.transform(X_test_sub).toarray())
    y_train.extend(y_train_sub)
    y_test.extend(y_test_sub)

# この後の学習を高速化するためにスパースマトリクスに戻す
X_train = csr_matrix(X_train)
X_test = csr_matrix(X_test)
```

```
<class 'scipy.sparse._csr.csr_matrix'>
<class 'scipy.sparse._csr.csr_matrix'>
<class 'scipy.sparse._csr.csr_matrix'>
<class 'scipy.sparse._csr.csr_matrix'>
```

In [ ]:
```
model=svm.SVC()
model.fit(X_train,y_train)
```

Out[ ]:
SVC
SVC()

In [ ]:
```
pred=model.predict(X_test)
print('正答率:',accuracy_score(y_test,pred))
cm=confusion_matrix(y_test,pred)
print(cm)
```

```
正答率: 0.4175
[[97  2  0  1]
 [71 27  1  1]
 [81  0 18  1]
 [73  1  1 25]]
```

集英社のデータばかりのデータセットで学習したために，そのバランスに過学習してしまい，
性能の良くないモデルになってしまっていることがわかる．

3-4. 深層学習の基礎と展望 - ディープニューラルネットワーク（DNN） / 3-6. 予測・判断 - ROC曲線、AUC（Area Under the Curve)

メディア芸術データベースのデータを利用して，簡単な分類を行うディープニューラルネットワークのサンプルコードを見ていこう．

今回は，データベース上からゲームパッケージを取得し，「タイトル」「発行者」「ゲームプラットフォーム」を特徴量化して対象年齢（CERO）がいずれに該当するものなのかを予測する多クラス分類モデルを作っていく．

データの取得・整形

メディア芸術データベース・ラボ上で提供されているSPARQLクエリサービスを利用して，ゲームパッケージのデータを取得していく．

以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる．

取得したCSVファイルを「ゲームパッケージ.csv」として保存し，このノートブック上にアップロードしよう．

※「3-4. 学習用データと学習済みモデル」，「3-6. 決定木」，「3-6. サポートベクターマシン」，「3-7. 形態素解析」，「3-7. かな漢字変換」，「3-7. 表現学習」と同じデータセットを作成するため，もし同じものを持っている場合は以下の取得作業は不要なので，そちらをアップロードしよう．

```
PREFIX ma: <https://mediaarts-db.bunka.go.jp/data/property#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?発行者 ?プラットフォーム ?レーティング ?公開年月日
WHERE {
  ?アイテム
        schema:genre "ゲームパッケージ";
        schema:contentRating ?レーティング;
        schema:gamePlatform ?プラットフォーム;
        schema:publisher ?発行者;
        rdfs:label ?ラベル;
        ma:datePublished ?公開年月日.
}
```

取得したデータセットを，Python上で扱いやすい形にしていく． pandasというライブラリを用いて，csv形式のファイルの中身をDataFrameという型に変換し，扱っていく．

In [1]:
```
import pandas as pd
```

In [2]:
```
games = pd.read_csv("/content/ゲームパッケージ.csv")
games = games.drop_duplicates()
games = games.reset_index(drop=True)
games
```

Out[2]:

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
0	Dance Dance Revolution GB2	コナミ株式会社	ゲームボーイ	CERO D (17才以上対象)	2000-11-16
1	プロ野球 ファミスタ リターンズ	株式会社バンダイナムコエンターテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2015-10-08
2	プロ野球 ファミスタ 2011	株式会社バンダイナムコゲームス	ニンテンドー3DS	CERO A (全年齢対象)	2011-03-31
3	フロッガー3D	株式会社コナミデジタルエンタテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2011-09-22
4	ブレイブリーセカンド	株式会社スクウェア・エニックス	ニンテンドー3DS	CERO C (15才以上対象)	2015-04-23
...	...	...	...	...	...
21976	バイオショック インフィニット PlayStation Now版	テイクツー・インタラクティブ・ジャパン合同会社	プレイステーション Now	CERO D (17才以上対象)	2017-09-19
21977	バイオハザード リベレーションズ アンベールド エディション PlayStation Now版	株式会社カプコン	プレイステーション Now	CERO D (17才以上対象)	2016-09-20
21978	巫剣神威控 PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO C (15才以上対象)	2017-07-20
21979	ネバーエンディング ナイトメア PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO D (17才以上対象)	2017-07-20
21980	雷電IV OverKill PlayStation Now版	モス	プレイステーション Now	CERO A (全年齢対象)	2017-02-21

21981 rows × 5 columns

まず，前処理前のデータの取得したgamesの["レーティング"]カラム（対象年齢を表す列）には，どんなものがいくつ含まれているか見てみよう．

In [3]:
```
import collections
```

In [4]:
```
collections.Counter(games["レーティング"]) #games["レーティング"]内に何がどれだけあるか数える
```

Out[4]:
```
Counter({'CERO D (17才以上対象)': 1966,
         'CERO A (全年齢対象)': 9232,
         'CERO C (15才以上対象)': 3237,
         'CERO B (12才以上対象)': 4706,
         'CERO Z (18才以上のみ対象)': 752,
         'IARC 16+': 2,
         'IARC 12+': 4,
         'IARC 3+': 4,
         'CERO 全年齢': 852,
```

```
        'CERO 12': 131,
        'CERO 15': 85,
        'CERO 18': 41,
        '推奨年齢 全年齢': 610,
        '推奨年齢 18才以上': 50,
        'X指定 18才以上': 25,
        '推奨年齢 年齢制限 18才以上': 29,
        'ESRB E': 3,
        'PG': 1,
        '18': 4,
        '審査済証 映像倫理機構 ETHICS 成人指定 18歳未満販売・貸出禁止': 12,
        'ELSPA 3+': 1,
        'ESRB T': 7,
        'ESRB M': 4,
        'EOCS 審査済 一般ソフトウェア General Softwere 15歳以上推奨 (コンピュータソフトウェア倫理機構)':
4,
        '18才未満お断り (コンピュータソフトウェア倫理機構)': 112,
        'EOCS 審査済 (コンピュータソフトウェア倫理機構)': 26,
        'EOCS 審査済 一般ソフトウェア General Softwere 12歳以上推奨 (コンピュータソフトウェア倫理機構)':
3,
        '18+': 1,
        '12+': 5,
        'USK 16': 1,
        '一般 (15歳以上推奨)': 4,
        '3+': 5,
        '3': 1,
        '12': 2,
        '16+': 4,
        '16': 2,
        'USK ab 18': 4,
        'USK ab 12': 1,
        '16歳未満不適': 4,
        'E 一般向': 44})
```

想定よりもさまざまなデータが混在している．特に，CERO+[英字]のものとCERO+[数字or日本語]のものはそれぞれ対象がかぶっていそうなデータであるため扱いが難しそうだ．

これについてCEROの公式サイトによれば，

> 年齢区分マーク （平成18年3月1日改訂）

とのことだったので，今回は2006年以降の新レーティングである「CERO　A/B/C/D/Z」のみを対象として分類を行なっていくこととする．

`games`に取得していたデータの中から，「CERO A/B/C/D/Z」のデータのみを取得して新たに`games`に保存していく．

In [5]:
```
games = games.query(
    "レーティング == 'CERO A (全年齢対象)' or レーティング == 'CERO B (12才以上対象)' or レーティング
)
```

データは揃ったが，このままタイトルや発行者などの文字列をDNNモデルに直接入力することはできない．

そこで，今回特徴量にする情報群をベクトル化していく．

In [6]:
```
# 必要なモジュールの読み込み
!pip install mecab-python3
```

```
!pip install unidic-lite
import MeCab

import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

```
Collecting mecab-python3
  Downloading mecab_python3-1.0.8-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl (581 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 581.7/58
1.7 kB 6.9 MB/s eta 0:00:00
Installing collected packages: mecab-python3
Successfully installed mecab-python3-1.0.8
Collecting unidic-lite
  Downloading unidic-lite-1.0.8.tar.gz (47.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.4/47.
4 MB 17.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: unidic-lite
  Building wheel for unidic-lite (setup.py) ... done
  Created wheel for unidic-lite: filename=unidic_lite-1.0.8-py3-none-any.whl size=47658817 sha
256=ab32972735ac2439c55e4ca68fe966055a2b4391fee9d44d22912479b0685e90
  Stored in directory: /root/.cache/pip/wheels/89/e8/68/f9ac36b8cc6c8b3c96888cd57434ab
ed96595d444f42243853
Successfully built unidic-lite
Installing collected packages: unidic-lite
Successfully installed unidic-lite-1.0.8
```

タイトルのベクトル化

文章を単語に分かち書きし，それら単語の出現の有無を0,1ビットで表現するベクトルに変換する手法を取ることにする．

In [7]:
```
# 分かち書きを行うインスタンスwakatiの作成
wakati = MeCab.Tagger('-Owakati')

# 分かち書きしたタイトルを格納するリスト
wakati_titles = []

for title in games['ラベル']:
  wakati_titles.append(wakati.parse(title).strip())

# 文字列->ベクトルを変換するインスタンスの作成・学習
tokenizer = Tokenizer()
tokenizer.fit_on_texts(wakati_titles)

# どの単語がタイトル内に現れているか?を0,1で表すベクトルに変換し,labelsに格納
labels = tokenizer.texts_to_matrix(wakati_titles, mode="binary")
```

発行者のベクトル化

各会社をカテゴリ変数と見做し，one-hotベクトル表現によってこれを表すことにする．

In [8]:
```
# ワンホットベクトルへのエンコードを行うインスタンスonehot_encoderの作成
onehot_encoder = OneHotEncoder(sparse = False,dtype = int)
```

```
# 発行者がどこか?を表すワンホットベクトルへの学習・変換を同時に行う関数.変換後はcreatorsに格納
creators = onehot_encoder.fit_transform(np.array(games['発行者']).reshape(-1,1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarni
ng: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse
_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

プラットフォームのベクトル化

発行者の時と同様，各プラットフォームをカテゴリ変数と見做し，one-hotベクトル表現によってこれを表すことにする.

In [9]:
```
# プラットフォームが何か?を表すワンホットベクトルへの学習・変換を同時に行う関数.変換後はplatformsに格
platforms = onehot_encoder.fit_transform(np.array(games['プラットフォーム']).reshape(-1,1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarni
ng: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse
_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

特徴量ベクトルの作成

In [10]:
```
# 各特徴量を横並びに結合して,1つのゲームパッケージにつき1つの結合された特徴量ベクトルが対応するよう
X = np.concatenate([labels, creators,platforms], axis=1)
print(f"データの数: {X.shape[0]}, 1つのデータを表す特徴量ベクトルの次元数: {X.shape[1]}")
```

```
データの数: 19893, 1つのデータを表す特徴量ベクトルの次元数: 13606
```

正解ラベル（対象年齢）のベクトル化

こちらも今まで同様，各対象年齢をカテゴリ変数と見做し，one-hotベクトル表現によってこれを表すことにする.

In [11]:
```
y = onehot_encoder.fit_transform(np.array(games['レーティング']).reshape(-1,1))
print(f"データの数: {y.shape[0]}, 正解ベクトルの次元数: {y.shape[1]}")
```

```
データの数: 19893, 正解ベクトルの次元数: 5
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarni
ng: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse
_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

上記の特徴量ベクトルに対応する5次元の正解ベクトルが作成された.

学習データとテストデータに分ける

In [12]:
```
# デフォルトでは学習データは75%,テストデータは25%になる
X_train, X_test, y_train, y_test = train_test_split(np.array(X), np.array(y), shuffle=True)
```

ディープニューラルネットワーク（DNN）

上記で用意したデータセットを利用して，「全結合層」「活性化層」「ドロップアウト層」のみを用いた簡単な多クラス分類モデルを作っていく．（モデルの学習には10分ほどかかるので注意）

In [13]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
```

In [14]:
```
# モデルの構築
model = Sequential()
model.add(Dense(256, input_shape=(len(X_train[0]),))) #256次元の全結合層
model.add(Activation('relu')) #活性化
model.add(Dropout(0.2)) #ドロップアウト
model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dense(len(y_train[0])))# 出力層の次元数は,正解データの次元数(=5次元)に合わせる
model.add(Activation('softmax'))

model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

# 学習
model.fit(X_train, y_train, validation_split=0.1, epochs=10, batch_size=256)

# 評価
result = model.evaluate(X_test, y_test)
print(f"Testデータに対するLoss: {result[0]}, Testデータに対するAccuracy: {result[1]}")
```

```
Epoch 1/10
53/53 [==============================] - 8s 137ms/step - loss: 1.1553 - accuracy: 0.
5560 - val_loss: 0.7959 - val_accuracy: 0.7078
Epoch 2/10
53/53 [==============================] - 6s 108ms/step - loss: 0.5372 - accuracy: 0.
8230 - val_loss: 0.5181 - val_accuracy: 0.8204
Epoch 3/10
53/53 [==============================] - 6s 122ms/step - loss: 0.2393 - accuracy: 0.
9302 - val_loss: 0.4493 - val_accuracy: 0.8458
Epoch 4/10
53/53 [==============================] - 6s 104ms/step - loss: 0.1221 - accuracy: 0.
9660 - val_loss: 0.4401 - val_accuracy: 0.8499
Epoch 5/10
53/53 [==============================] - 6s 121ms/step - loss: 0.0740 - accuracy: 0.
9810 - val_loss: 0.4597 - val_accuracy: 0.8586
Epoch 6/10
53/53 [==============================] - 6s 106ms/step - loss: 0.0489 - accuracy: 0.
9870 - val_loss: 0.4880 - val_accuracy: 0.8539
Epoch 7/10
53/53 [==============================] - 6s 121ms/step - loss: 0.0375 - accuracy: 0.
9899 - val_loss: 0.5107 - val_accuracy: 0.8552
Epoch 8/10
53/53 [==============================] - 5s 96ms/step - loss: 0.0290 - accuracy: 0.9
917 - val_loss: 0.5469 - val_accuracy: 0.8579
Epoch 9/10
53/53 [==============================] - 6s 107ms/step - loss: 0.0236 - accuracy: 0.
9931 - val_loss: 0.5485 - val_accuracy: 0.8606
Epoch 10/10
53/53 [==============================] - 6s 110ms/step - loss: 0.0196 - accuracy: 0.
9934 - val_loss: 0.5672 - val_accuracy: 0.8579
156/156 [==============================] - 1s 8ms/step - loss: 0.5954 - accuracy: 0.
8673
Testデータに対するLoss: 0.595369279384613, Testデータに対するAccuracy: 0.8673099875450
134
```

3-6. 予測・判断 - ROC曲線、AUC（Area Under the Curve）

ROC曲線とは偽陽性率(=1−Specificity)を横軸，再現率（Recall）を縦軸にとって陽性/陰性の判断基準を変えたときの変化を表す曲線のことである． また，AUCとはROC曲線とx軸の間の面積で，値が1に近いほど予測モデルの性能が高いと判断できるものである（ランダムな分類モデルの場合，ROC曲線は直線y=xに近づき，AUCの値は0.5程度になる）．

多クラス分類問題においてもAUCの計算およびROC曲線のプロットができるが，陽性/陰性の判断の仕方や平均の仕方を工夫する必要がある．いくつか方法はあるが，以下に2つ例を挙げる．

1. マクロ平均
    - クラスごとにROC曲線を描き，それぞれに対してAUCを計算した後で単純な平均を取る
    - クラスごとに属する要素の個数によらず，すべてのクラスの結果を平等に反映させたい場合こちらが適している

    A. One vs Rest（OvR）
        - ある特定のクラスを正解としたとき，その他のクラス全てを不正解とみなす
        - ROC曲線は1つのクラスにつき1つ描くことができる

    B. One vs One（OvO）
        - ある特定のクラスを正解としたとき，その他のクラス全てとの組み合わせでペアを作り，個々のペアに対してROC曲線を描く
        - ROC曲線はペアの作り方*2通り描くことができる
2. マイクロ平均
    - クラスごとの「実際に陽性のデータを陽性と予測できた数（TP）」と「実際は陰性のデータを陽性と誤予測した数（FP）」を合算してモデルのTP，FPとすることで1つのROCを描き，AUCを計算する
    - 要素数が少ないクラスの重要度が低い場合こちらが適している

今回はマイクロ平均を採用することとする．

In [15]:
```
# 必要なモジュールのインポート
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.metrics import RocCurveDisplay
```

In [16]:
```
# 予測データを用意
y_pred = model.predict(X_test)

# 形を確認
y_pred.shape
```

```
156/156 [==============================] - 1s 8ms/step
```

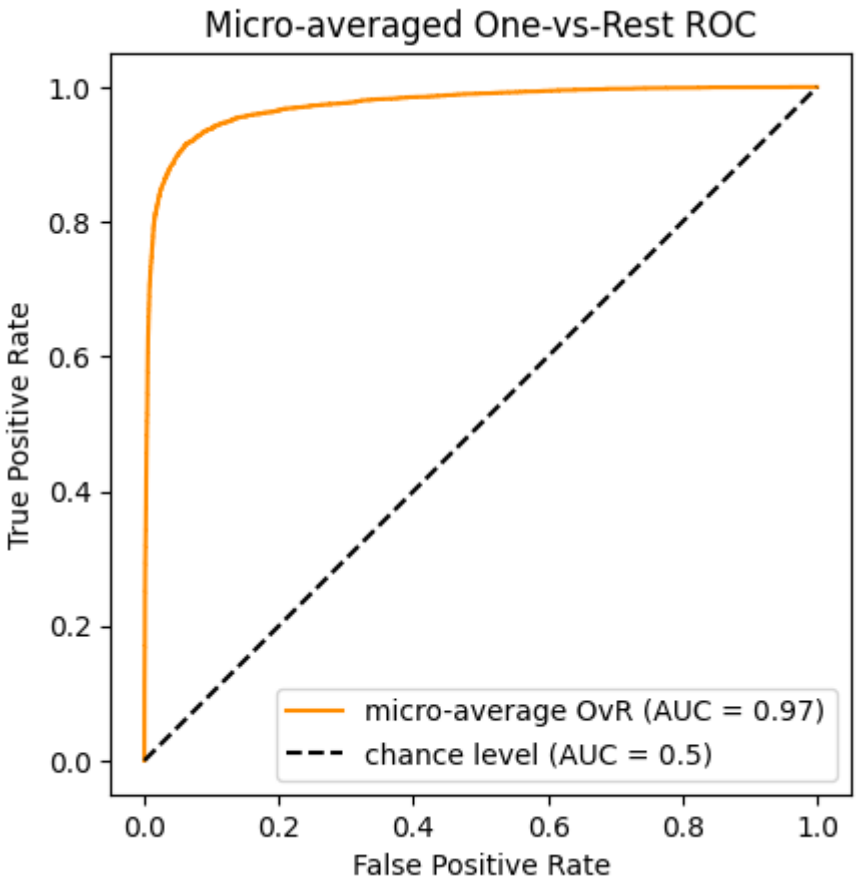
Out[16]:
```
(4974, 5)
```

In [17]:
```
# 予測データを一次元に平坦化
y_pred_ravel = y_pred.ravel()
print(y_pred_ravel.shape)
```

```
# 正解データも平坦化
y_test_ravel = y_test.ravel()
print(y_test_ravel.shape)
```

```
(24870,)
(24870,)
```

In [18]:
```
# ROC曲線のプロット
RocCurveDisplay.from_predictions(
    y_test_ravel,
    y_pred_ravel,
    name="micro-average OvR",
    color="darkorange",
)
plt.plot([0, 1], [0, 1], "k--", label="chance level (AUC = 0.5)")
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Micro-averaged One-vs-Rest ROC")
plt.legend()
plt.show()
```



In [19]:
```
# AUCの値はROC曲線のプロットにも記載されるが,計算のみであればはこちらでもできる
micro_roc_auc_ovr = roc_auc_score(
    y_test,
    y_pred,
    multi_class="ovr",
    average="micro",
)

print(f"Micro-averaged One-vs-Rest ROC AUC score: {micro_roc_auc_ovr:.2f}")
```

```
Micro-averaged One-vs-Rest ROC AUC score: 0.97
```

AUCの値から見ても，かなり精度のいい分類モデルができたと言える.

3-4. 深層学習の基礎と展望 - 学習用データと学習済みモデル

単語をベクトル化する有名な手法としてWord2Vecというものがある．これは単語を類似度に紐づくベクトルに埋め込むことができるという面白いモデルである．これは目的に応じて自分でコーパス（学習用の文章を大量に集めたもの）を用意してモデルの構築・学習を行うこともできるが，いちからモデルを学習しようとすると膨大な量のコーパスが必要になる．

あまり大きなコーパスが用意できない場合には，事前学習済みのモデルに対して追加で学習を行い，パラメータを微調整することで対応することもできる．

本節では，ゲームパッケージのタイトル群に登場する単語を学習済みモデルに追加で学習させることで，少ない学習量でもゲームタイトル的に近い意味を持つ単語が出力されるようにしよう．

学習用データの作成

メディア芸術データベース・ラボ上で提供されているSPARQLクエリサービスを利用して，ゲームパッケージのデータを取得していく．

以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる．

取得したCSVファイルを「ゲームパッケージ.csv」として保存し，このノートブック上にアップロードしよう．

※「3-4. ディープニューラルネットワーク（DNN）」，「3-6. 決定木」，「3-6. サポートベクターマシン」，「3-7. 形態素解析」，「3-7. かな漢字変換」，「3-7. 表現学習」と同じデータセットを作成するため，もし同じものを持っている場合は以下の取得作業は不要なので，そちらをアップロードしよう．

```
PREFIX ma: <https://mediaarts-db.bunka.go.jp/data/property#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?発行者 ?プラットフォーム ?レーティング ?公開年月日
WHERE {
  ?アイテム
        schema:genre "ゲームパッケージ";
        schema:contentRating ?レーティング;
        schema:gamePlatform ?プラットフォーム;
        schema:publisher ?発行者;
        rdfs:label ?ラベル;
        ma:datePublished ?公開年月日.
}
```

取得したデータセットを，Python上で扱いやすい形にしていく．pandasというライブラリを用いて，csv形式のファイルの中身をDataFrameという型に変換し，扱っていく．

In [1]:
```
import pandas as pd
games = pd.read_csv("/content/ゲームパッケージ.csv")
```

```
games = games.drop_duplicates()
games = games.reset_index(drop=True)
games
```

Out[1]:

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
0	Dance Dance Revolution GB2	コナミ株式会社	ゲームボーイ	CERO D (17才以上対象)	2000-11-16
1	プロ野球 ファミスタ リターンズ	株式会社バンダイナムコエンターテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2015-10-08
2	プロ野球 ファミスタ 2011	株式会社バンダイナムコゲームス	ニンテンドー3DS	CERO A (全年齢対象)	2011-03-31
3	フロッガー3D	株式会社コナミデジタルエンタテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2011-09-22
4	ブレイブリーセカンド	株式会社スクウェア・エニックス	ニンテンドー3DS	CERO C (15才以上対象)	2015-04-23
...	...	...	...	...	...
21976	バイオショック インフィニット PlayStation Now版	テイクツー・インタラクティブ・ジャパン合同会社	プレイステーション Now	CERO D (17才以上対象)	2017-09-19
21977	バイオハザード リベレーションズ アンベールド エディション PlayStation Now版	株式会社カプコン	プレイステーション Now	CERO D (17才以上対象)	2016-09-20
21978	巫剣神威控 PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO C (15才以上対象)	2017-07-20
21979	ネバーエンディング ナイトメア PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO D (17才以上対象)	2017-07-20
21980	雷電IV OverKill PlayStation Now版	モス	プレイステーション Now	CERO A (全年齢対象)	2017-02-21

21981 rows × 5 columns

ゲームパッケージのタイトルを，単語の分かち書きにする．

In [2]:

```
# 必要なモジュールの読み込み
!pip install mecab-python3
!pip install unidic-lite
import MeCab
import re

# 分かち書きを行うインスタンスwakatiの作成
wakati = MeCab.Tagger('-Owakati')

# 分かち書きしたタイトルを格納するリスト
wakati_titles = []
```

```
for title in games['ラベル']:
  wakati_titles.append(wakati.parse(title).strip())

# w2vに追加学習させるコーパス
new_corpus = []
for wakati_title in wakati_titles:
  tmp = re.sub("\u3000|\|\?", "", wakati_title).split(" ")
  new_corpus.append([token for token in tmp if len(token) > 1]) # 単語長が2以上のもののみを学習
```

```
Collecting mecab-python3
  Downloading mecab_python3-1.0.8-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl (581 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 581.7/58
1.7 kB 11.7 MB/s eta 0:00:00 0:00:01
Installing collected packages: mecab-python3
Successfully installed mecab-python3-1.0.8
Collecting unidic-lite
  Downloading unidic-lite-1.0.8.tar.gz (47.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.4/47.
4 MB 16.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: unidic-lite
  Building wheel for unidic-lite (setup.py) ... done
  Created wheel for unidic-lite: filename=unidic_lite-1.0.8-py3-none-any.whl size=47658817 sha
256=a9a9a224d466416d551af5910ab035945d18dfc1ea28aab045cacd1a943e59b2
  Stored in directory: /root/.cache/pip/wheels/89/e8/68/f9ac36b8cc6c8b3c96888cd57434ab
ed96595d444f42243853
Successfully built unidic-lite
Installing collected packages: unidic-lite
Successfully installed unidic-lite-1.0.8
```

In [3]:
```
print(new_corpus[0])
```

```
['Dance', 'Dance', 'Revolution', 'GB']
```

学習済みモデルのロード

In [4]:
```
#必要なモジュールの読み込み
!pip install --upgrade gensim
from gensim.models import word2vec

# 事前学習済みモデルのインストール
!wget "http://public.shiroyagi.s3.amazonaws.com/latest-ja-word2vec-gensim-model.zip"
!unzip "/content/latest-ja-word2vec-gensim-model.zip"
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages (4.3.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from
gensim) (1.25.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from ge
nsim) (1.11.4)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.10/dist-packages (f
rom gensim) (6.4.0)
--2024-02-14 05:58:56--  http://public.shiroyagi.s3.amazonaws.com/latest-ja-word2vec-gensi
m-model.zip
Resolving public.shiroyagi.s3.amazonaws.com (public.shiroyagi.s3.amazonaws.com)... 52.219.13
6.205, 52.219.8.173, 52.219.16.213, ...
Connecting to public.shiroyagi.s3.amazonaws.com (public.shiroyagi.s3.amazonaws.com)|52.219.
136.205|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 132936751 (127M) [application/zip]
```

```
Saving to: ‘latest-ja-word2vec-gensim-model.zip’

latest-ja-word2vec- 100%[===================>] 126.78M  19.5MB/s    in 7.4s

2024-02-14 05:59:04 (17.0 MB/s) - ‘latest-ja-word2vec-gensim-model.zip’ saved [132936751/132936751]

Archive:  /content/latest-ja-word2vec-gensim-model.zip
  inflating: word2vec.gensim.model
  inflating: word2vec.gensim.model.syn1neg.npy
  inflating: word2vec.gensim.model.wv.syn0.npy
```

In [5]:
```
# 事前学習済みモデルのロード
w2v_origin = word2vec.Word2Vec.load("/content/word2vec.gensim.model")

# こちらを追加学習用モデルにしていく
w2v = word2vec.Word2Vec.load("/content/word2vec.gensim.model")
```

In [6]:
```
# 新しいコーパスを追加で学習させる
w2v.build_vocab(new_corpus, update=True)
w2v.train(new_corpus, total_examples=w2v.corpus_count, epochs=w2v.epochs)
```

Out[6]:
```
(227781, 521520)
```

In [7]:
```
# 学習前の「DS」に近い単語
print(w2v_origin.wv.most_similar("DS"))
```

```
[('PS2', 0.9018317461013794), ('PS3', 0.8976770639419556), ('PSP', 0.8939802050590515), ('GBA', 0.8769130706787109), ('PS4', 0.8751301169395447), ('SFC', 0.8686517477035522), ('3DS', 0.8658976554870605), ('PS Vita', 0.8649856448173523), ('ゲームキューブ', 0.8592252731323242), ('パワーアップキット', 0.8539624810218811)]
```

In [8]:
```
# 学習後の「DS」に近い単語
print(w2v.wv.most_similar("DS"))
```

```
[('DSi', 0.9567910432815552), ('ポータブル', 0.9450507164001465), ('DX', 0.9403018951416016), ('ウェア', 0.9384950399398804), ('Wii', 0.9258646965026855), ('パズル', 0.9234594106674194), ('ウォーズ', 0.9170921444892883), ('アドベンチャー', 0.9161989092826843), ('セレクション', 0.9157466292381287), ('EX', 0.9150858521461487)]
```

学習の前後で，近しい単語の種類が変わっていることがわかる．

元々のモデル上では，ゲームプラットフォームという意味において"DS"に近しいものが出力されている．

それに対しゲームタイトルをコーパスとして追加で学習したものでは，ゲームタイトルに（オマケ的に）よく登場する単語として"DS"に似たようなものが出力されている．

このように学習済みモデルを利用することで，自然言語処理としてはあまり大規模ではなかった `wakati_titles` のみでも直感的に意味の通る結果を得ることができる．

3-4. 深層学習の基礎と展望 - 再帰型ニューラルネットワーク（RNN）

メディア芸術データベース・ラボ上のデータを利用して，簡単な予測を行う再帰型ニューラルネットワークのサンプルコードを見ていこう．

データの取得・整形

今回は，データベース上からマンガ雑誌を取得し，週刊少年ジャンプの価格遷移を予想するモデルを作っていく． 以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる．

取得したCSVファイルを「週刊少年ジャンプ.csv」として保存し，このノートブック上にアップロードしよう．

※3-4. 畳み込みニューラルネットワーク（CNN）と同じデータセットを作成するため，もし同じものを持っている場合は以下の取得作業は不要なので，そちらをアップロードしよう．

```
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?公開年月日 ?価格 ?ページ
WHERE {
  ?アイテム
        schema:genre "マンガ雑誌単号";
        schema:name "週刊少年ジャンプ";
        rdfs:label ?ラベル;
        schema:price ?価格;
        schema:numberOfPages ?ページ;
        schema:datePublished ?公開年月日.
}
```

In [ ]:
```
# 必要なモジュールの読み込み
import pandas as pd
import numpy as np
import datetime as dt
import re
import matplotlib.pyplot as plt
```

In [ ]:
```
jumps = pd.read_csv("/content/週刊少年ジャンプ.csv")
jumps['公開年月日'] = pd.to_datetime(jumps['公開年月日'])
jumps.sort_values(by = '公開年月日', ascending = True, inplace = True)
jumps.reset_index(inplace=True, drop=True)

for i in jumps.index:
  price = re.sub(r"\D", "", jumps["価格"][i])
  if price != "":
    jumps["価格"][i] = int(price)
  else:
    jumps["価格"][i] = None

jumps = jumps.dropna()
jumps.reset_index(inplace=True, drop=True)
jumps
```

```
<ipython-input-2-0559c6bca7de>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  jumps["価格"][i] = int(price)
<ipython-input-2-0559c6bca7de>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  jumps["価格"][i] = None
```

Out[ ]:

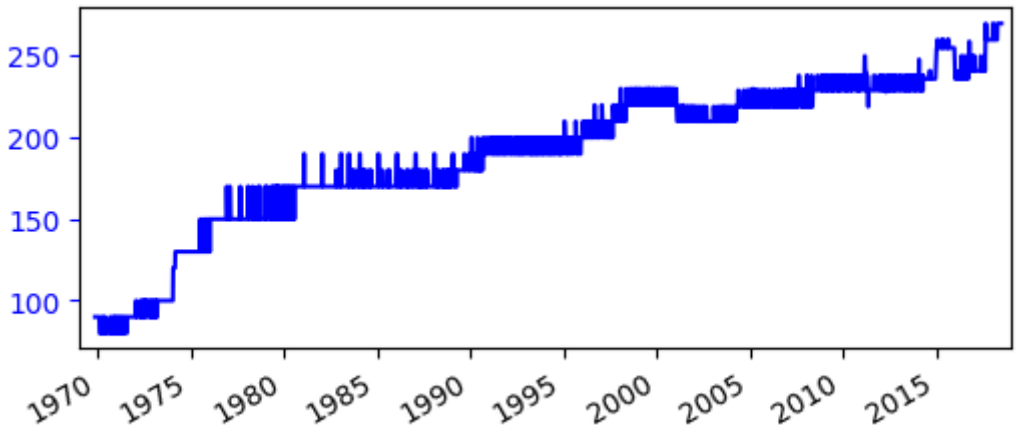
	ラベル	公開年月日	価格	ページ
0	週刊少年ジャンプ 1969年 表示号数20	1969-11-03	90	296
1	週刊少年ジャンプ 1969年 表示号数21	1969-11-10	90	294
2	週刊少年ジャンプ 1969年 表示号数22	1969-11-17	90	312
3	週刊少年ジャンプ 1969年 表示号数23	1969-11-24	90	294
4	週刊少年ジャンプ 1969年 表示号数24	1969-12-01	90	294
...	...	...	...	...
2380	週刊少年ジャンプ 2018年 表示号数24	2018-05-28	270	464p
2381	週刊少年ジャンプ 2018年 表示号数25	2018-06-04	270	496p
2382	週刊少年ジャンプ 2018年 表示号数26	2018-06-11	270	450p
2383	週刊少年ジャンプ 2018年 表示号数27	2018-06-18	270	492p
2384	週刊少年ジャンプ 2018年 表示号数28	2018-06-25	270	494p

2385 rows × 4 columns

得られた価格の推移を見てみよう.

In [ ]:
```
span = pd.to_datetime(["1969-01-01", "2019-01-01"])

fig, ax = plt.subplots(figsize=(6, 2.5))
fig.autofmt_xdate()

ax.plot(jumps["公開年月日"], jumps["価格"], color='b')
ax.tick_params('y', colors='b')

ax.set_xlim(span)
```

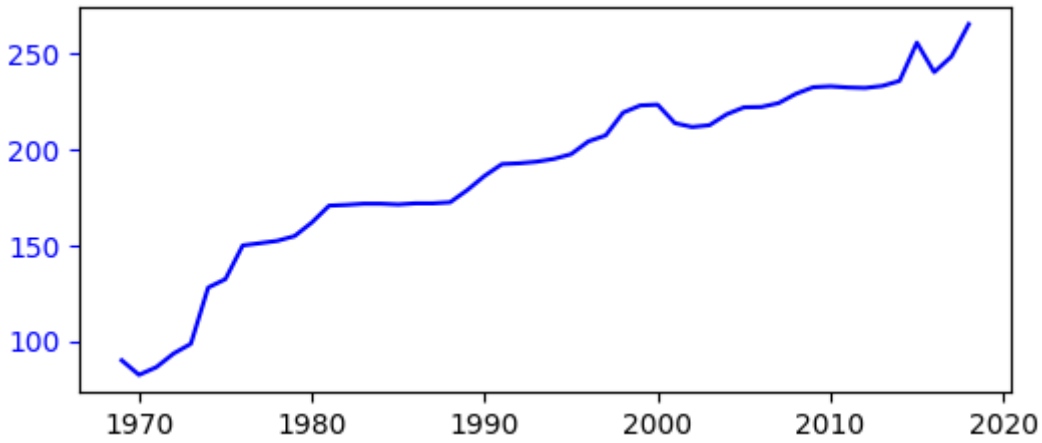
Out[ ]:
```
(-365.0, 17897.0)
```



かなり周期の小さい変動（振動）に加えて，全体的には増加の傾向の変動が見られることがわかる．

この二つの変動の種類が混ざっていると予測が難しそうなので，簡易化のために年価格平均のグラフを考えることで，全体的な増加の傾向を読み取りやすいグラフにしよう．

In [ ]:
```
years = []
price_means = []
for year in range(1969,2019):
  data = jumps[(jumps['公開年月日'] >= dt.datetime(year,1,1)) & (jumps['公開年月日'] < dt.datet
  price_mean = sum(data)/len(data.index)
  years.append(year)
  price_means.append(price_mean)

fig, ax = plt.subplots(figsize=(6, 2.5))
ax.plot(years, price_means, color='b')
ax.tick_params('y', colors='b')
```



価格平均の推移を見てみると，発売当初や最近のものについては変動が大きく予測が難しそうなので，今回はすべてのデータではなく，1985年〜2015年までの30年間のデータを利用することとする．

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, SimpleRNN
```

In [ ]:
```
price = pd.DataFrame(price_means, index=years, columns=["価格平均"])

# 1975-2015までのデータに絞る
price_1975_2015 = price[(price.index >=1975) & (price.index <= 2015)]
price_1975_2015.reset_index(inplace=True, drop=True)
```

```
# 1975-2015までの価格データを,0~1の間に入るように正規化
min_price = min(price_1975_2015["価格平均"])
price_width = max(price_1975_2015["価格平均"]) - min_price
price_1975_2015["価格平均"] = pd.Series([(price - min_price)/price_width for price in price_19

# 次の価格を予測するために,過去10件分の推移をみる
sequence_len = 10

# Xの1要素は連続10件分の価格が入ったリストになる
X = []
# y[i]には,X[i]の10件の価格推移の後の価格のスカラー値(=予測すべき値)が入る
y = []


for i in range(len(price_1975_2015.index)-sequence_len):
  X.append(list(price_1975_2015["価格平均"][i:i+sequence_len]))
  y.append(price_1975_2015["価格平均"][i+sequence_len])

X = np.array(X)
y = np.array(y)


# 直近10個のデータをテストデータ用に残しておく
num_test = 10

X_train_rnn = np.array(X[:-num_test])
y_train_rnn = np.array(y[:-num_test])
X_test_rnn = np.array(X[-num_test:])
y_test_rnn = np.array(y[-num_test:])


# 入力Xと正解データyの準備
# 入力を(サンプル数、予測に使うデータの数(今回はsequence_len = 10)、データの説明変数の数(今回は
X_train_rnn = X_train_rnn.reshape((X_train_rnn.shape[0], X_train_rnn.shape[1], 1))
y_train_rnn = y_train_rnn.reshape((y_train_rnn.shape[0], 1, 1))

X_test_rnn = X_test_rnn.reshape((X_test_rnn.shape[0], X_test_rnn.shape[1], 1))
y_test_rnn = y_test_rnn.reshape((y_test_rnn.shape[0], 1, 1))

# モデルの構築

n_middle = 100 # RNNの中間層の次元数

model_rnn = Sequential()
model_rnn.add(SimpleRNN(n_middle, input_shape=(sequence_len, 1), return_sequences=False))
model_rnn.add(Dense(1))
model_rnn.add(Activation('linear'))

# モデルのコンパイル
model_rnn.compile(loss='mean_absolute_error', optimizer="adam")

# モデルの学習
model_rnn.fit(X_train_rnn, y_train_rnn, validation_split=0.05, epochs=150, batch_size=1)

# モデルの評価
model_rnn.evaluate(X_test_rnn, y_test_rnn)

test_rnn = X_train_rnn[-1]

# 予測値を使ってさらに予測していくことで5年分の予測を行う
for i in range(num_test):
  input = test_rnn[-sequence_len:]
  test_rnn = np.append(test_rnn, model_rnn.predict(input)[0][-1]).reshape(-1,1)  # 最後から{se
```

```
y_pred = test_rnn[sequence_len:]
plt.clf()
fig_rnn, ax_rnn = plt.subplots(figsize=(12, 5))

# 予測値を青,実際の値を赤でプロット
# 前半は予測のために用いる実測値なので,後半のみが変わる
ax_rnn.plot(test_rnn.reshape(-1), color='b')
ax_rnn.plot(list(price_1975_2015["価格平均"][-(num_test+sequence_len+1):]), color='r')
```

```
<ipython-input-6-237fbd43b7d4>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  price_1975_2015["価格平均"] = pd.Series([(price - min_price)/price_width for price in price_1975_2015["価格平均"]])
Epoch 1/150
19/19 [==============================] - 2s 28ms/step - loss: 0.1785 - val_loss: 0.2467
Epoch 2/150
19/19 [==============================] - 0s 9ms/step - loss: 0.1051 - val_loss: 0.1478
Epoch 3/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0881 - val_loss: 0.0739
Epoch 4/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0935 - val_loss: 0.0236
Epoch 5/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0578 - val_loss: 0.0400
Epoch 6/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0650 - val_loss: 0.0357
Epoch 7/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0453 - val_loss: 0.0414
Epoch 8/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0458 - val_loss: 0.1027
Epoch 9/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0509 - val_loss: 0.0821
Epoch 10/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0620 - val_loss: 0.0743
Epoch 11/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0542 - val_loss: 0.0295
Epoch 12/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0513 - val_loss: 0.0123
Epoch 13/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0590 - val_loss: 0.1405
Epoch 14/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0494 - val_loss: 0.1135
```

```
Epoch 15/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0550 - val_loss: 0.08
93
Epoch 16/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0721 - val_loss: 0.07
78
Epoch 17/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0365 - val_loss: 0.04
66
Epoch 18/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0397 - val_loss: 0.05
45
Epoch 19/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0354 - val_loss: 0.07
44
Epoch 20/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0546 - val_loss: 0.02
17
Epoch 21/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0455 - val_loss: 0.04
45
Epoch 22/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0356 - val_loss: 0.01
00
Epoch 23/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0502 - val_loss: 0.03
90
Epoch 24/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0466 - val_loss: 0.01
06
Epoch 25/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0406 - val_loss: 0.08
29
Epoch 26/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0393 - val_loss: 0.01
47
Epoch 27/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0323 - val_loss: 0.01
24
Epoch 28/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0344 - val_loss: 0.03
38
Epoch 29/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0299 - val_loss: 0.03
15
Epoch 30/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0379 - val_loss: 0.06
63
Epoch 31/150
19/19 [==============================] - 0s 4ms/step - loss: 0.0502 - val_loss: 0.03
26
Epoch 32/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0294 - val_loss: 0.01
11
Epoch 33/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0301 - val_loss: 0.01
75
Epoch 34/150
```

19/19 [==============================] - 0s 6ms/step - loss: 0.0296 - val_loss: 0.01
74
Epoch 35/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0298 - val_loss: 0.06
26
Epoch 36/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0491 - val_loss: 0.01
30
Epoch 37/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0354 - val_loss: 0.01
68
Epoch 38/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0363 - val_loss: 0.02
78
Epoch 39/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0446 - val_loss: 0.06
92
Epoch 40/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0294 - val_loss: 0.03
22
Epoch 41/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0356 - val_loss: 0.01
76
Epoch 42/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0339 - val_loss: 0.03
28
Epoch 43/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0280 - val_loss: 0.01
69
Epoch 44/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0272 - val_loss: 0.04
34
Epoch 45/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0285 - val_loss: 0.01
15
Epoch 46/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0341 - val_loss: 0.02
47
Epoch 47/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0299 - val_loss: 0.02
09
Epoch 48/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0303 - val_loss: 0.01
11
Epoch 49/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0376 - val_loss: 0.03
23
Epoch 50/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0268 - val_loss: 0.04
80
Epoch 51/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0276 - val_loss: 0.04
73
Epoch 52/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0230 - val_loss: 0.03
49
Epoch 53/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0274 - val_loss: 0.06

25
Epoch 54/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0308 - val_loss: 0.01
16
Epoch 55/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0236 - val_loss: 0.03
52
Epoch 56/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0264 - val_loss: 0.01
17
Epoch 57/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0393 - val_loss: 0.01
24
Epoch 58/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0223 - val_loss: 0.02
09
Epoch 59/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0245 - val_loss: 0.01
55
Epoch 60/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0237 - val_loss: 0.02
38
Epoch 61/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0286 - val_loss: 0.01
16
Epoch 62/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0302 - val_loss: 0.02
77
Epoch 63/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0217 - val_loss: 0.02
82
Epoch 64/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0269 - val_loss: 0.01
03
Epoch 65/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0309 - val_loss: 0.01
92
Epoch 66/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0279 - val_loss: 0.03
19
Epoch 67/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0292 - val_loss: 0.00
98
Epoch 68/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0267 - val_loss: 0.04
98
Epoch 69/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0275 - val_loss: 0.03
58
Epoch 70/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0251 - val_loss: 0.01
30
Epoch 71/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0287 - val_loss: 0.02
13
Epoch 72/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0253 - val_loss: 0.00
95

```
Epoch 73/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0240 - val_loss: 0.03
86
Epoch 74/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0181 - val_loss: 0.02
45
Epoch 75/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0248 - val_loss: 0.03
41
Epoch 76/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0249 - val_loss: 0.05
90
Epoch 77/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0225 - val_loss: 0.02
21
Epoch 78/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0309 - val_loss: 0.04
71
Epoch 79/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0253 - val_loss: 0.03
92
Epoch 80/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0248 - val_loss: 0.00
42
Epoch 81/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0246 - val_loss: 0.02
60
Epoch 82/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0231 - val_loss: 0.01
00
Epoch 83/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0228 - val_loss: 0.01
00
Epoch 84/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0254 - val_loss: 0.04
70
Epoch 85/150
19/19 [==============================] - 0s 4ms/step - loss: 0.0230 - val_loss: 0.03
50
Epoch 86/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0317 - val_loss: 0.02
49
Epoch 87/150
19/19 [==============================] - 0s 4ms/step - loss: 0.0225 - val_loss: 0.01
74
Epoch 88/150
19/19 [==============================] - 0s 4ms/step - loss: 0.0217 - val_loss: 0.00
40
Epoch 89/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0359 - val_loss: 0.04
51
Epoch 90/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0207 - val_loss: 0.00
77
Epoch 91/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0209 - val_loss: 0.04
79
Epoch 92/150
```

19/19 [==============================] - 0s 6ms/step - loss: 0.0214 - val_loss: 0.00
71
Epoch 93/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0185 - val_loss: 0.01
40
Epoch 94/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0385 - val_loss: 0.07
06
Epoch 95/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0270 - val_loss: 0.02
03
Epoch 96/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0241 - val_loss: 0.02
72
Epoch 97/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0336 - val_loss: 0.01
14
Epoch 98/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0247 - val_loss: 0.01
09
Epoch 99/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0263 - val_loss: 0.01
82
Epoch 100/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0195 - val_loss: 0.00
69
Epoch 101/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0272 - val_loss: 0.02
14
Epoch 102/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0205 - val_loss: 0.00
36
Epoch 103/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0253 - val_loss: 0.02
51
Epoch 104/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0211 - val_loss: 0.03
72
Epoch 105/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0191 - val_loss: 0.01
54
Epoch 106/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0222 - val_loss: 0.04
70
Epoch 107/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0275 - val_loss: 0.05
75
Epoch 108/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0208 - val_loss: 0.00
97
Epoch 109/150
19/19 [==============================] - 0s 9ms/step - loss: 0.0182 - val_loss: 0.00
13
Epoch 110/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0232 - val_loss: 0.00
86
Epoch 111/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0242 - val_loss: 0.02

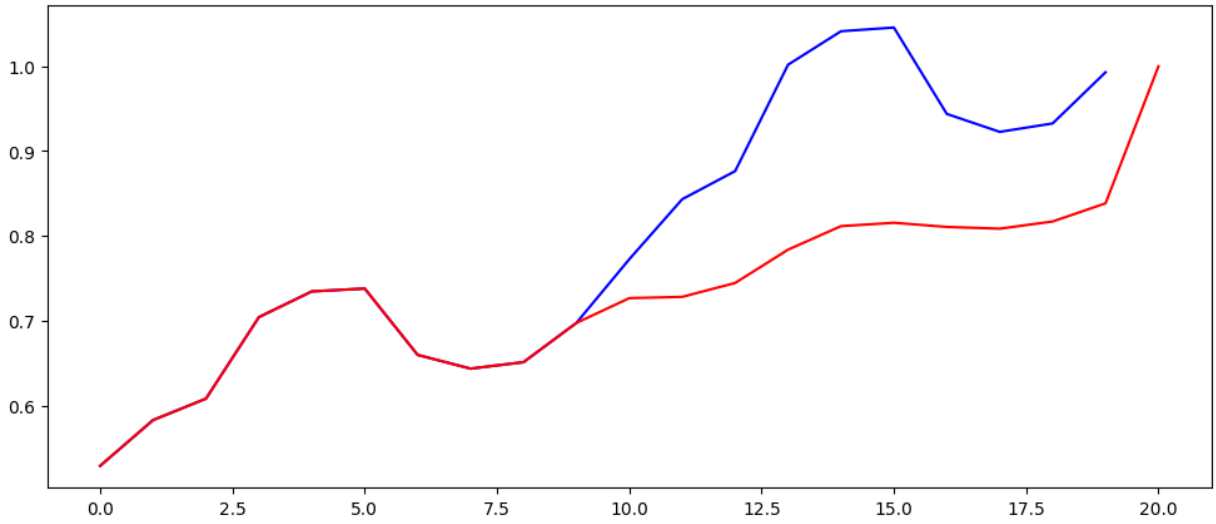
```
46
Epoch 112/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0221 - val_loss: 0.01
09
Epoch 113/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0243 - val_loss: 0.04
79
Epoch 114/150
19/19 [==============================] - 0s 9ms/step - loss: 0.0205 - val_loss: 0.03
45
Epoch 115/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0195 - val_loss: 0.00
55
Epoch 116/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0179 - val_loss: 0.01
45
Epoch 117/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0189 - val_loss: 0.05
05
Epoch 118/150
19/19 [==============================] - 0s 7ms/step - loss: 0.0177 - val_loss: 0.03
41
Epoch 119/150
19/19 [==============================] - 0s 8ms/step - loss: 0.0202 - val_loss: 0.05
73
Epoch 120/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0242 - val_loss: 0.03
36
Epoch 121/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0192 - val_loss: 0.00
51
Epoch 122/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0200 - val_loss: 0.03
14
Epoch 123/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0227 - val_loss: 0.01
55
Epoch 124/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0227 - val_loss: 0.00
32
Epoch 125/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0198 - val_loss: 0.00
96
Epoch 126/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0213 - val_loss: 0.00
45
Epoch 127/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0212 - val_loss: 0.01
56
Epoch 128/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0234 - val_loss: 0.00
44
Epoch 129/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0211 - val_loss: 0.04
68
Epoch 130/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0228 - val_loss: 0.03
11
```

```
Epoch 131/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0202 - val_loss: 0.03
13
Epoch 132/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0192 - val_loss: 0.03
18
Epoch 133/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0214 - val_loss: 0.04
55
Epoch 134/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0188 - val_loss: 0.00
68
Epoch 135/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0233 - val_loss: 0.01
17
Epoch 136/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0225 - val_loss: 0.01
11
Epoch 137/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0303 - val_loss: 0.01
10
Epoch 138/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0215 - val_loss: 0.01
63
Epoch 139/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0175 - val_loss: 0.04
75
Epoch 140/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0213 - val_loss: 0.02
99
Epoch 141/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0211 - val_loss: 0.02
35
Epoch 142/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0214 - val_loss: 0.08
72
Epoch 143/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0303 - val_loss: 0.01
77
Epoch 144/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0214 - val_loss: 0.00
58
Epoch 145/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0244 - val_loss: 0.02
45
Epoch 146/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0204 - val_loss: 0.01
63
Epoch 147/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0191 - val_loss: 0.03
21
Epoch 148/150
19/19 [==============================] - 0s 6ms/step - loss: 0.0198 - val_loss: 0.01
54
Epoch 149/150
19/19 [==============================] - 0s 5ms/step - loss: 0.0221 - val_loss: 0.01
09
Epoch 150/150
```

```
19/19 [==============================] - 0s 5ms/step - loss: 0.0230 - val_loss: 0.00
83
1/1 [==============================] - 0s 277ms/step - loss: 0.0210
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 23ms/step
```

Out[ ]: [<matplotlib.lines.Line2D at 0x797757772650>]

<Figure size 640x480 with 0 Axes>



グラフを見比べることで，増加の傾向を学習できていることがわかる．

3-4. 深層学習の基礎と展望 - 畳み込みニューラルネットワーク（CNN）

メディア芸術データベース・ラボ上のサービスを利用して取得したデータを用いて，簡単な予測を行う畳み込みニューラルネットワークのサンプルコードを見ていこう．

データの取得・整形

今回は，データベース上からマンガ雑誌を取得し，週刊少年ジャンプの価格遷移を予想するモデルを作っていく． 以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる．

取得したCSVファイルを「週刊少年ジャンプ.csv」として保存し，このノートブック上にアップロードしよう．

※3-4. 再帰型ニューラルネットワーク（RNN）と同じデータセットを作成するため，もし同じものを持っている場合は以下の取得作業は不要なので，そちらをアップロードしよう．

```
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?公開年月日 ?価格 ?ページ
WHERE {
  ?アイテム
        schema:genre "マンガ雑誌単号";
        schema:name "週刊少年ジャンプ";
        rdfs:label ?ラベル;
        schema:price ?価格;
        schema:numberOfPages ?ページ;
        schema:datePublished ?公開年月日.
}
```

In [ ]:
```
# 必要なモジュールの読み込み
import pandas as pd
import numpy as np
import datetime as dt
import re
import matplotlib.pyplot as plt
```

In [ ]:
```
jumps = pd.read_csv("/content/週刊少年ジャンプ.csv")
jumps = jumps.drop_duplicates().reset_index(drop=True)
jumps['公開年月日'] = pd.to_datetime(jumps['公開年月日'])
jumps.sort_values(by = '公開年月日', ascending = True, inplace = True)
jumps.reset_index(inplace=True, drop=True)

for i in jumps.index:
  price = re.sub(r"\D", "", jumps["価格"][i])
  if price != "":
    jumps["価格"][i] = int(price)
  else:
    jumps["価格"][i] = None

jumps = jumps.dropna()
```

```
jumps.reset_index(inplace=True, drop=True)
jumps
```

```
<ipython-input-2-ab251b28bf1f>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 jumps["価格"][i] = int(price)
<ipython-input-2-ab251b28bf1f>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 jumps["価格"][i] = None
```

Out[ ]:

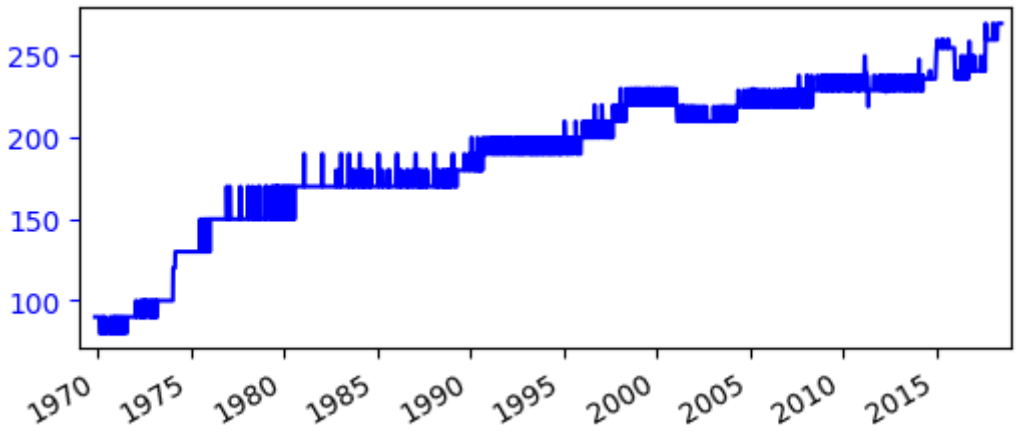
	ラベル	公開年月日	価格	ページ
0	週刊少年ジャンプ 1969年 表示号数20	1969-11-03	90	296
1	週刊少年ジャンプ 1969年 表示号数21	1969-11-10	90	294
2	週刊少年ジャンプ 1969年 表示号数22	1969-11-17	90	312
3	週刊少年ジャンプ 1969年 表示号数23	1969-11-24	90	294
4	週刊少年ジャンプ 1969年 表示号数24	1969-12-01	90	294
...	...	...	...	...
2379	週刊少年ジャンプ 2018年 表示号数24	2018-05-28	270	464p
2380	週刊少年ジャンプ 2018年 表示号数25	2018-06-04	270	496p
2381	週刊少年ジャンプ 2018年 表示号数26	2018-06-11	270	450p
2382	週刊少年ジャンプ 2018年 表示号数27	2018-06-18	270	492p
2383	週刊少年ジャンプ 2018年 表示号数28	2018-06-25	270	494p

2384 rows × 4 columns

得られた価格の推移を見てみよう.

In [ ]:
```
span = pd.to_datetime(["1969-01-01", "2019-01-01"])

fig, ax = plt.subplots(figsize=(6, 2.5))
fig.autofmt_xdate()

ax.plot(jumps["公開年月日"], jumps["価格"], color='b')
ax.tick_params('y', colors='b')

ax.set_xlim(span)
```

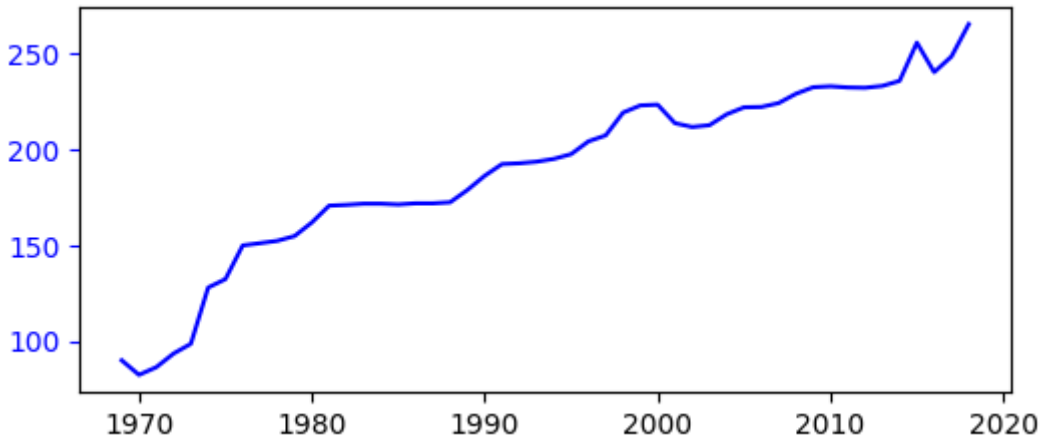
Out[ ]:
```
(-365.0, 17897.0)
```



かなり周期の小さい変動（振動）に加えて，全体的には増加の傾向の変動が見られることがわかる．

この二つの変動の種類が混ざっていると予測が難しそうなので，簡易化のために年価格平均のグラフを考えることで，全体的な増加の傾向を読み取りやすいグラフにしよう．

In [ ]:
```
years = []
price_means = []
for year in range(1969,2019):
  data = jumps[(jumps['公開年月日'] >= dt.datetime(year,1,1)) & (jumps['公開年月日'] < dt.datet
  price_mean = sum(data)/len(data.index)
  years.append(year)
  price_means.append(price_mean)

fig, ax = plt.subplots(figsize=(6, 2.5))
ax.plot(years, price_means, color='b')
ax.tick_params('y', colors='b')
```



価格平均の推移を見てみると，発売当初や最近のものについては変動が大きく予測が難しそうなので，今回はすべてのデータではなく，1985年〜2015年までの30年間のデータを利用することとする．

In [ ]:
```
#必要なモジュールの読み込み
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Reshape, Activation
from tensorflow.keras.optimizers import Adam
```

In [ ]:
```
price = pd.DataFrame(price_means, index=years, columns=["価格平均"])

# 1975-2015までのデータに絞る
price_1975_2015 = price[(price.index >=1975) & (price.index <= 2015)]
```

```
price_1975_2015.reset_index(inplace=True, drop=True)

# 1975-2015までの価格データを,0~1の間に入るように正規化
min_price = min(price_1975_2015["価格平均"])
price_width = max(price_1975_2015["価格平均"]) - min_price
price_1975_2015["価格平均"] = pd.Series([(price - min_price)/price_width for price in price_19

#次の価格を予測するために,過去16件分の推移をみて4件分の予測をすることを考える
num_for_predict = 16
num_to_predict = 4

# Xは,1要素に連続16件分の価格が入ったリストになる
X = []
# yは,y[i]には,X[i]の16件の価格推移の後の4件分の価格が入ったリストになる
y = []


for i in range(len(price_1975_2015.index)-num_for_predict-num_to_predict+1):
  X.append(list(price_1975_2015["価格平均"][i : i+num_for_predict]))
  y.append(price_1975_2015["価格平均"][i+num_for_predict : i+num_for_predict+num_to_pre

X = np.array(X) # (データ数, 16)
y = np.array(y) # (データ数,4)


# 3回*4個=直近16個のデータをテストデータ用に残しておく
num_test = 3
X_train_cnn = np.array(X[:-num_test*num_to_predict])
y_train_cnn = np.array(y[:-num_test*num_to_predict])
X_test_cnn = np.array(X[-num_test*num_to_predict:])
y_test_cnn = np.array(y[-num_test*num_to_predict:])


print(price_1975_2015.shape)

# 入力Xと正解データyの準備
# 入力を(サンプル数、学習に使うデータの数(16)、データの説明変数の数(今回は価格のみの1次元))の形に
X_train_cnn = X_train_cnn.reshape((X_train_cnn.shape[0], X_train_cnn.shape[1], 1))
y_train_cnn = y_train_cnn.reshape((y_train_cnn.shape[0], y_train_cnn.shape[1], 1))

X_test_cnn = X_test_cnn.reshape((X_test_cnn.shape[0], X_test_cnn.shape[1], 1))
y_test_cnn = y_test_cnn.reshape((y_test_cnn.shape[0], y_test_cnn.shape[1], 1))

# モデルの構築
model_cnn = Sequential()
model_cnn.add(Conv1D(filters=16, kernel_size=8, padding='same', activation='relu', strides=1,
model_cnn.add(MaxPooling1D(pool_size=2)) # 上の16個のフィルタによってそれぞれ作成された16の
model_cnn.add(Conv1D(filters=8, kernel_size=8, padding='same', activation='relu', strides=1))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Conv1D(filters=1, kernel_size=8, padding='same', activation='relu', strides=1))
model_cnn.add(Activation('linear'))

# モデルのコンパイル
model_cnn.compile(loss='mean_absolute_error', optimizer="adam")

print(model_cnn.summary())

# モデルの学習
model_cnn.fit(X_train_cnn, y_train_cnn, validation_split=0.05, epochs=100, batch_size=1)

# モデルの評価
model_cnn.evaluate(X_test_cnn, y_test_cnn)

test_cnn = X_train_cnn[-1].reshape(1, num_for_predict, 1) # shape: (batch, num_for_predict,
```

```
# 予測値を使ってさらに予測...を繰り返す
for i in range(num_test):
  input = test_cnn[:, -num_for_predict:, :]
  test_cnn = np.concatenate([test_cnn, model_cnn.predict(input)], axis=1) # 最後から{num_for_

y_pred = test_cnn[num_for_predict:]
plt.clf()
fig_cnn, ax_cnn = plt.subplots(figsize=(12, 5))

# 予測値を青,実際の値を赤でプロット
# 前半は予測のために用いる実測値なので,後半のみが変わる
ax_cnn.plot(test_cnn.reshape(-1), color='b')
ax_cnn.plot(list(price_1975_2015["価格平均"][-((num_test+1)*num_to_predict+num_for_pre
```

```
(41, 1)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 16, 16)            144

 max_pooling1d (MaxPooling1  (None, 8, 16)             0
 D)

 conv1d_1 (Conv1D)           (None, 8, 8)              1032

 max_pooling1d_1 (MaxPoolin  (None, 4, 8)              0
 g1D)

 conv1d_2 (Conv1D)           (None, 4, 1)              65

 activation (Activation)     (None, 4, 1)              0

=================================================================
Total params: 1241 (4.85 KB)
Trainable params: 1241 (4.85 KB)
Non-trainable params: 0 (0.00 Byte)
<ipython-input-6-fd04c636971f>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gui
de/indexing.html#returning-a-view-versus-a-copy
  price_1975_2015["価格平均"] = pd.Series([(price - min_price)/price_width for price in price_1
975_2015["価格平均"]]) # 全部で41件のデータがある
_________________________________________________________________
None
Epoch 1/100
9/9 [==============================] - 1s 22ms/step - loss: 0.5872 - val_loss: 0.621
1
Epoch 2/100
9/9 [==============================] - 0s 4ms/step - loss: 0.5421 - val_loss: 0.5525
Epoch 3/100
9/9 [==============================] - 0s 4ms/step - loss: 0.4643 - val_loss: 0.4295
Epoch 4/100
9/9 [==============================] - 0s 4ms/step - loss: 0.3454 - val_loss: 0.2519
Epoch 5/100
9/9 [==============================] - 0s 4ms/step - loss: 0.2354 - val_loss: 0.2321
```

```
Epoch 6/100
9/9 [==============================] - 0s 4ms/step - loss: 0.2025 - val_loss: 0.1645
Epoch 7/100
9/9 [==============================] - 0s 4ms/step - loss: 0.1595 - val_loss: 0.1103
Epoch 8/100
9/9 [==============================] - 0s 4ms/step - loss: 0.1220 - val_loss: 0.0758
Epoch 9/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0926 - val_loss: 0.0929
Epoch 10/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0712 - val_loss: 0.0757
Epoch 11/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0618 - val_loss: 0.0788
Epoch 12/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0585 - val_loss: 0.0935
Epoch 13/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0500 - val_loss: 0.0924
Epoch 14/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0454 - val_loss: 0.1003
Epoch 15/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0410 - val_loss: 0.0959
Epoch 16/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0395 - val_loss: 0.0945
Epoch 17/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0370 - val_loss: 0.1013
Epoch 18/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0380 - val_loss: 0.0878
Epoch 19/100
9/9 [==============================] - 0s 6ms/step - loss: 0.0355 - val_loss: 0.0903
Epoch 20/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0361 - val_loss: 0.0887
Epoch 21/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0361 - val_loss: 0.0932
Epoch 22/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0421 - val_loss: 0.0970
Epoch 23/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0378 - val_loss: 0.0801
Epoch 24/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0382 - val_loss: 0.1126
Epoch 25/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0389 - val_loss: 0.0903
Epoch 26/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0383 - val_loss: 0.1105
Epoch 27/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0370 - val_loss: 0.0944
Epoch 28/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0340 - val_loss: 0.0924
Epoch 29/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0354 - val_loss: 0.0984
Epoch 30/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0354 - val_loss: 0.0952
Epoch 31/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0348 - val_loss: 0.0919
Epoch 32/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0341 - val_loss: 0.0979
Epoch 33/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0370 - val_loss: 0.1000
Epoch 34/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0355 - val_loss: 0.0947
```

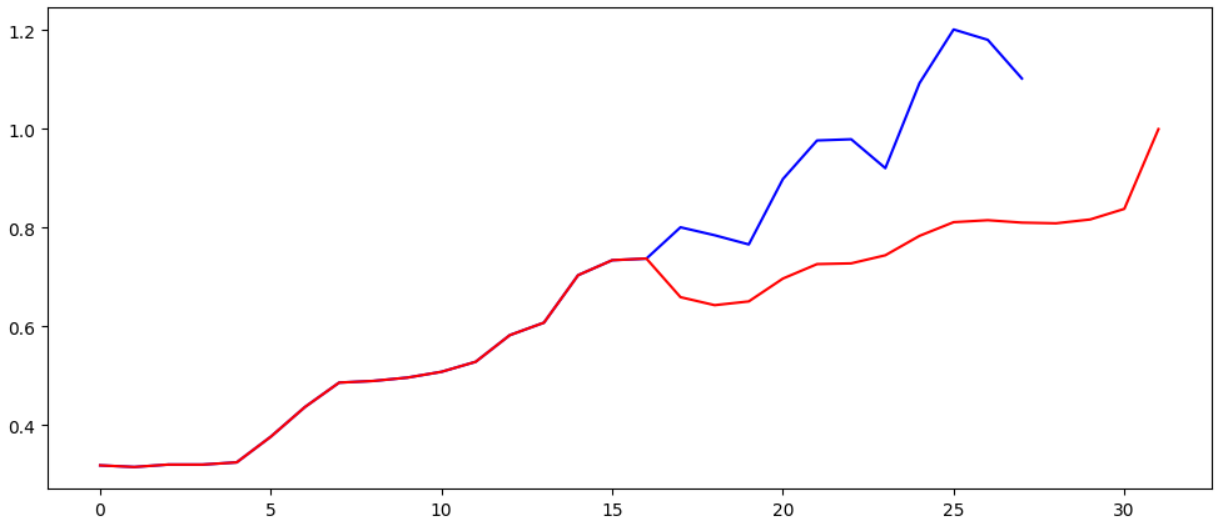
```
Epoch 35/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0363 - val_loss: 0.1069
Epoch 36/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0357 - val_loss: 0.0902
Epoch 37/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0352 - val_loss: 0.0996
Epoch 38/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0328 - val_loss: 0.0964
Epoch 39/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0325 - val_loss: 0.1013
Epoch 40/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0329 - val_loss: 0.0929
Epoch 41/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0329 - val_loss: 0.0953
Epoch 42/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0358 - val_loss: 0.1024
Epoch 43/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0347 - val_loss: 0.0915
Epoch 44/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0330 - val_loss: 0.0975
Epoch 45/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0311 - val_loss: 0.1065
Epoch 46/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0344 - val_loss: 0.1024
Epoch 47/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0320 - val_loss: 0.0945
Epoch 48/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0327 - val_loss: 0.1066
Epoch 49/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0305 - val_loss: 0.0975
Epoch 50/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0325 - val_loss: 0.1045
Epoch 51/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0308 - val_loss: 0.0965
Epoch 52/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0327 - val_loss: 0.1058
Epoch 53/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0321 - val_loss: 0.1002
Epoch 54/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0341 - val_loss: 0.1059
Epoch 55/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0311 - val_loss: 0.0941
Epoch 56/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0302 - val_loss: 0.1132
Epoch 57/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0367 - val_loss: 0.1146
Epoch 58/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0396 - val_loss: 0.0915
Epoch 59/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0308 - val_loss: 0.1088
Epoch 60/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0313 - val_loss: 0.1008
Epoch 61/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0304 - val_loss: 0.1043
Epoch 62/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0301 - val_loss: 0.1038
Epoch 63/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0324 - val_loss: 0.1048
```

```
Epoch 64/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0317 - val_loss: 0.0996
Epoch 65/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0290 - val_loss: 0.1076
Epoch 66/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0313 - val_loss: 0.0946
Epoch 67/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0316 - val_loss: 0.1122
Epoch 68/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0311 - val_loss: 0.0978
Epoch 69/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0315 - val_loss: 0.1125
Epoch 70/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0310 - val_loss: 0.1037
Epoch 71/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0306 - val_loss: 0.1038
Epoch 72/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0404 - val_loss: 0.1253
Epoch 73/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0342 - val_loss: 0.0869
Epoch 74/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0337 - val_loss: 0.1273
Epoch 75/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0283 - val_loss: 0.0958
Epoch 76/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0296 - val_loss: 0.1135
Epoch 77/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0302 - val_loss: 0.0946
Epoch 78/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0309 - val_loss: 0.0984
Epoch 79/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0290 - val_loss: 0.1066
Epoch 80/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0301 - val_loss: 0.1085
Epoch 81/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0290 - val_loss: 0.1016
Epoch 82/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0297 - val_loss: 0.1077
Epoch 83/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0294 - val_loss: 0.0991
Epoch 84/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0285 - val_loss: 0.1034
Epoch 85/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0298 - val_loss: 0.0979
Epoch 86/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0292 - val_loss: 0.1024
Epoch 87/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0288 - val_loss: 0.1104
Epoch 88/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0324 - val_loss: 0.0935
Epoch 89/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0334 - val_loss: 0.1386
Epoch 90/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0329 - val_loss: 0.1055
Epoch 91/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0279 - val_loss: 0.1031
Epoch 92/100
9/9 [==============================] - 0s 6ms/step - loss: 0.0287 - val_loss: 0.1084
```

```
Epoch 93/100
9/9 [==============================] - 0s 6ms/step - loss: 0.0286 - val_loss: 0.1079
Epoch 94/100
9/9 [==============================] - 0s 6ms/step - loss: 0.0294 - val_loss: 0.1008
Epoch 95/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0276 - val_loss: 0.1127
Epoch 96/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0279 - val_loss: 0.0981
Epoch 97/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0281 - val_loss: 0.1103
Epoch 98/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0294 - val_loss: 0.1035
Epoch 99/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0274 - val_loss: 0.1080
Epoch 100/100
9/9 [==============================] - 0s 5ms/step - loss: 0.0282 - val_loss: 0.0997
1/1 [==============================] - 0s 111ms/step - loss: 0.2107
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
```

Out[ ]: [<matplotlib.lines.Line2D at 0x79061ce04c10>]

<Figure size 640x480 with 0 Axes>



グラフを見比べることで，増加の傾向を学習できていることがわかる．

3-6. 予測・判断 - MSE（Mean Square Error）、特徴量の効果的な選択

3-4.や3-6.で扱ってきた正解率や適合率などの指標は分類モデルの評価指標であったが，回帰モデル（すなわち，カテゴリではなく数値として予測が行えるモデル）に対してはMSE（Mean Square Error，平均二乗誤差）という評価指標が存在する．

ここでは「週刊少年ジャンプのページ数・価格・公開年月日から消費者物価指数（書籍）を予測する回帰モデル」と，「なんの関連もないデータを使って学習した同様の予測器」のMSEを比べることでモデルの精度を比較してみよう．

週刊少年ジャンプのデータについては3-4. 畳み込みニューラルネットワーク（CNN）/再帰型ニューラルネットワーク（RNN）のデータと同様のものを扱う．持っていない場合は以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる．

取得したCSVファイルを「週刊少年ジャンプ.csv」として保存し，このノートブック上にアップロードしよう．

```
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?公開年月日 ?価格 ?ページ
WHERE {
  ?アイテム
        schema:genre "マンガ雑誌単号";
        schema:name "週刊少年ジャンプ";
        rdfs:label ?ラベル;
        schema:price ?価格;
        schema:numberOfPages ?ページ;
        schema:datePublished ?公開年月日.
}
```

消費者物価指数（書籍）は政府統計の総合窓口e-statのDB上で情報を絞ってダウンロードしてくることができる．整形済みのものを用意したので，こちらから取得し，「消費者物価指数_全国_書籍.csv」としてアップロードしよう．

データの整形

In [ ]:
```
# 必要なモジュールの読み込み
import pandas as pd
import numpy as np
import re
import datetime as dt
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
# ジャンプのデータ(説明変数)
jumps = pd.read_csv("/content/週刊少年ジャンプ.csv")
jumps = jumps.drop_duplicates().reset_index(drop=True)

jumps['公開年月日'] = pd.to_datetime(jumps['公開年月日'])
jumps.sort_values(by = '公開年月日', ascending = True, inplace = True)
```

```
jumps.reset_index(inplace=True, drop=True)

for i in jumps.index:
  price = re.sub(r"\D", "", jumps["価格"][i])
  if price != "":
    jumps["価格"][i] = int(price)
  else:
    jumps["価格"][i] = None

  page = re.sub(r"\D", "", jumps["ページ"][i])
  if page != "":
    jumps["ページ"][i] = int(page)
  else:
    jumps["ページ"][i] = None

jumps = jumps.dropna()
jumps.reset_index(inplace=True, drop=True)
jumps
```

```
<ipython-input-6-fa9a78688f98>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 jumps["価格"][i] = int(price)
<ipython-input-6-fa9a78688f98>:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 jumps["ページ"][i] = int(page)
<ipython-input-6-fa9a78688f98>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 jumps["価格"][i] = None
```

Out[ ]:

	ラベル	公開年月日	価格	ページ
0	週刊少年ジャンプ 1969年 表示号数20	1969-11-03	90	296
1	週刊少年ジャンプ 1969年 表示号数21	1969-11-10	90	294
2	週刊少年ジャンプ 1969年 表示号数22	1969-11-17	90	312
3	週刊少年ジャンプ 1969年 表示号数23	1969-11-24	90	294
4	週刊少年ジャンプ 1969年 表示号数24	1969-12-01	90	294
...	...	...	...	...
2379	週刊少年ジャンプ 2018年 表示号数24	2018-05-28	270	464
2380	週刊少年ジャンプ 2018年 表示号数25	2018-06-04	270	496
2381	週刊少年ジャンプ 2018年 表示号数26	2018-06-11	270	450
2382	週刊少年ジャンプ 2018年 表示号数27	2018-06-18	270	492
2383	週刊少年ジャンプ 2018年 表示号数28	2018-06-25	270	494

2384 rows × 4 columns

In [ ]:

```
jumps_month = pd.DataFrame(columns=["公開年", "公開月", "価格", "ページ"])
dates = []
# 月平均にする
for year in range(1970, 2016):
  for month in range(1,13):
    data = jumps[(jumps['公開年月日'] >= dt.datetime(year,month,1)) & (jumps['公開年月日'] < dt
    price_mean = sum(data["価格"])/len(data.index)
    page_mean = sum(data["ページ"])/len(data.index)
    dates.append(dt.datetime(year, month, 1))
    series = pd.Series([year, month, price_mean, page_mean])
    jumps_month = pd.concat([jumps_month, pd.DataFrame(series.values.reshape(1, -1), columr
jumps_month.index = dates
jumps_month
```

Out[ ]:

	公開年	公開月	価格	ページ
1970-01-01	1970.0	1.0	90.00	327.333333

	公開年	公開月	価格	ページ
1970-02-01	1970.0	2.0	80.00	280.000000
1970-03-01	1970.0	3.0	82.00	282.400000
1970-04-01	1970.0	4.0	82.50	284.000000
1970-05-01	1970.0	5.0	82.50	284.000000
...	...	...	...	...
2015-08-01	2015.0	8.0	258.75	518.000000
2015-09-01	2015.0	9.0	255.00	492.000000
2015-10-01	2015.0	10.0	255.00	492.000000
2015-11-01	2015.0	11.0	255.00	488.400000
2015-12-01	2015.0	12.0	255.00	496.000000

552 rows × 4 columns

In [ ]:
```
# 消費者物価指数のデータ(目標変数)
cpis = pd.read_csv("/content/消費者物価指数_全国_書籍.csv", index_col=None).drop(columns=[
new_columns = [["物価指数"]]
new_index = []
for idx in cpis.index:
  year_month = re.findall(r"\d+", idx)
  if int(year_month[0]) < 2016:
    new_index.append(dt.datetime(int(year_month[0]), int(year_month[1]), 1))
  else:
    cpis = cpis.drop([idx], axis=0)

cpis.index = new_index
cpis.columns = new_columns
cpis = cpis.sort_index()
cpis
```

Out[ ]:

	物価指数
1970-01-01	17.9
1970-02-01	18.0
1970-03-01	18.0
1970-04-01	18.0
1970-05-01	18.0
...	...
2015-08-01	94.3
2015-09-01	94.1
2015-10-01	94.3
2015-11-01	94.4
2015-12-01	94.4

552 rows × 1 columns

In [ ]:
```
# ランダムな(意味のない)説明変数データ
dummy = pd.DataFrame(np.random.randint(0, 100, cpis.shape[0]), index = cpis.index, columns=
dummy
```

Out[ ]:

	ランダム
1970-01-01	9
1970-02-01	86
1970-03-01	90
1970-04-01	60
1970-05-01	58
...	...
2015-08-01	7
2015-09-01	2
2015-10-01	88
2015-11-01	29
2015-12-01	68

552 rows × 1 columns

In [ ]:
```
# 学習データとテストデータに分割
X_train = jumps_month[jumps_month.index < dt.datetime(2001,1,1)]
y_train = cpis[cpis.index < dt.datetime(2001,1,1)]
dummy_train = dummy[dummy.index < dt.datetime(2001,1,1)]

X_test = jumps_month[jumps_month.index > dt.datetime(2000,12,1)]
y_test = cpis[cpis.index > dt.datetime(2000,12,1)]
dummy_test = dummy[dummy.index > dt.datetime(2000,12,1)]

# 正規化
X_scaler = StandardScaler().fit(X_train)
X_train = X_scaler.transform(X_train)
X_test = X_scaler.transform(X_test)

dummy_scaler = StandardScaler().fit(dummy_train)
dummy_train = dummy_scaler.transform(dummy_train)
dummy_test = dummy_scaler.transform(dummy_test)
```

モデルの作成

In [ ]:
```
# 必要なモジュールのインポート
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [ ]:
```
# ジャンプのデータを使った予測モデルを作成
model = LinearRegression()
model.fit(X_train, y_train)

# ランダムデータを使った予測モデルを作成
model_dummy = LinearRegression()
model_dummy.fit(dummy_train, y_train)
```

Out[ ]:
LinearRegression
LinearRegression()

MSEによる評価

In [ ]:
```
# ジャンプのデータを使った予測モデルによるテストデータに対してのMSE
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)

# ランダムデータを使った予測モデルによるテストデータに対してのMSE
dummy_pred = model_dummy.predict(dummy_test)
mse_dummy = mean_squared_error(dummy_test, dummy_pred)

print("ジャンプのデータによるモデルのMSE: ", mse)
print("ランダムデータによるモデルのMSE: ", mse_dummy)
```

ジャンプのデータによるモデルのMSE: 328.8185725695346
ランダムデータによるモデルのMSE: 3409.641159688738

以上の結果から，ジャンプのデータを利用して計算したモデル方が適当な値で計算したモデルよりも良い結果（＝小さい損失）であったという評価ができる.

3-6. 予測・判断 - 特徴量の効果的な選択

線形回帰モデルにおいては，L1正則化項を用いた損失関数を用いる手法（Lasso回帰）で特徴量を選択することもできる.

Pythonではscikit-learnにLassoライブラリが用意されており，それを利用することで簡単に次元削減することができる.

In [ ]:
```
from sklearn.linear_model import Lasso
```

In [ ]:
```
lasso = Lasso()

# 学習
lasso.fit(X_train, y_train)

# 予測
y_pred = lasso.predict(X_test)

# MSE
mse_lasso = mean_squared_error(y_test, y_pred)
print("ジャンプのデータによるLasso回帰モデルのMSE: ", mse_lasso)

# 重要度の確認
lasso.coef_
```

ジャンプのデータによるLasso回帰モデルのMSE: 222.4534878864677

Out[ ]: array([12.94084538, 0. , 7.22576418, 0.])

MSEの値も通常の線形回帰モデルより小さく良い値になっている． また，重要度を見るといくつかの値（公開月，ページ）の重要度が0になっていることがわかる． このことから，公開年と価格の推移だけで十分な推測ができるということがわかる.

3-6. 予測・判断 - サポートベクターマシン（SVM）

サポートベクターマシンとは，データの代表点であるサポートベクタによって予測の識別境界を決定する手法で，異なるクラス間の距離（マージン）が最大になるように学習を行っているモデルである．その中でも，識別境界が直線で表される線形SVMや，カーネル関数を用いて曲線的な分類を可能にしたカーネルSVMなどが存在する．

メディア芸術データベース・ラボ上のサービスを利用して取得したデータを用いて，簡単な分類を行う線形SVMや，カーネルSVMのサンプルコードを見ていこう．

今回は，データベース上からゲームパッケージを取得し，「公開年月日」を特徴量化してプラットフォームがいずれに該当するものなのかを当てる多クラス分類モデルを作っていく．

データの取得・整形

メディア芸術データベース・ラボ上で提供されているSPARQLクエリサービスを利用して，ゲームパッケージのデータを取得していく．

以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる．

取得したCSVファイルを「ゲームパッケージ.csv」として保存し，このノートブック上にアップロードしよう．

※「3-4. ディープニューラルネットワーク（DNN）」，「3-4. 学習用データと学習済みモデル」，「3-6. 決定木」，「3-7. 形態素解析」，「3-7. かな漢字変換」，「3-7. 表現学習」と同じデータセットを作成するため，もし同じものを持っている場合は以下の取得作業は不要なので，そちらをアップロードしよう．

```
PREFIX ma: <https://mediaarts-db.bunka.go.jp/data/property#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?発行者 ?プラットフォーム ?レーティング ?公開年月日
WHERE {
  ?アイテム
        schema:genre "ゲームパッケージ";
        schema:contentRating ?レーティング;
        schema:gamePlatform ?プラットフォーム;
        schema:publisher ?発行者;
        rdfs:label ?ラベル;
        ma:datePublished ?公開年月日.
}
```

取得したデータセットを，Python上で扱いやすい形にしていく． pandasというライブラリを用いて，csv形式のファイルの中身をDataFrameという型に変換し，扱っていく．

In [1]:
```
# 必要なモジュールの読み込み
!pip install japanize-matplotlib
```

```
import numpy as np
import pandas as pd
import collections
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.datasets import make_blobs
from sklearn.svm import LinearSVC
import matplotlib.pyplot as plt
import japanize_matplotlib
```

```
Collecting japanize-matplotlib
  Downloading japanize-matplotlib-1.1.3.tar.gz (4.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.1/4.1 MB 32.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from japanize-matplotlib) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (4.48.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (1.25.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->japanize-matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->japanize-matplotlib) (1.16.0)
Building wheels for collected packages: japanize-matplotlib
  Building wheel for japanize-matplotlib (setup.py) ... done
  Created wheel for japanize-matplotlib: filename=japanize_matplotlib-1.1.3-py3-none-any.whl size=4120257 sha256=fee14390c6d942d1e12d1ebf07de575c340f75be36804af7b4463597ae728cd4
  Stored in directory: /root/.cache/pip/wheels/61/7a/6b/df1f79be9c59862525070e157e62b08eab8ece27c1b68fbb94
Successfully built japanize-matplotlib
Installing collected packages: japanize-matplotlib
Successfully installed japanize-matplotlib-1.1.3
```

In [2]:
```
games = pd.read_csv("/content/ゲームパッケージ.csv")
games = games.drop_duplicates()
games = games.reset_index(drop=True)
games
```

Out[2]:

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
0	Dance Dance Revolution GB2	コナミ株式会社	ゲームボーイ	CERO D (17才以上対象)	2000-11-16

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
1	プロ野球 ファミスタ リターンズ	株式会社バンダイナムコエンターテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2015-10-08
2	プロ野球 ファミスタ 2011	株式会社バンダイナムコゲームス	ニンテンドー3DS	CERO A (全年齢対象)	2011-03-31
3	フロッガー3D	株式会社コナミデジタルエンタテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2011-09-22
4	ブレイブリーセカンド	株式会社スクウェア・エニックス	ニンテンドー3DS	CERO C (15才以上対象)	2015-04-23
...	...	...	...	...	...
21976	バイオショック インフィニット PlayStation Now版	テイクツー・インタラクティブ・ジャパン合同会社	プレイステーション Now	CERO D (17才以上対象)	2017-09-19
21977	バイオハザード リベレーションズ アンベールド エディション PlayStation Now版	株式会社カプコン	プレイステーション Now	CERO D (17才以上対象)	2016-09-20
21978	巫剣神威控 PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO C (15才以上対象)	2017-07-20
21979	ネバーエンディング ナイトメア PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO D (17才以上対象)	2017-07-20
21980	雷電IV OverKill PlayStation Now版	モス	プレイステーション Now	CERO A (全年齢対象)	2017-02-21

21981 rows × 5 columns

データクレンジング

想定外の値を持つデータを除外する作業をする． 例えば，`games`の["公開年月日"]列には`yyyy-mm-dd`の形式で日付が入っていることが期待されるが，そうでない形式のものが含まれている可能性もある． 実際に見てみよう．

In [3]:
```
for day in games['公開年月日']:
  if len(day) < 10:
    print(day)
```

```
2007
2000
2009
2002
2006
2006
2006
2001
```

2014
2002
1999
1998
2007
2003
2008
2001
2003
2013
2017
2001
2002
2001
2005
1998
1998
2006

こういったものを先に除外しておくと，後の計算が楽に正しく行えるようになる．

In [4]:
```
games = games[games['公開年月日'].str.len() >= 10]
games = games.reset_index(drop=True)
games
```

Out[4]:

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
0	Dance Dance Revolution GB2	コナミ株式会社	ゲームボーイ	CERO D (17才以上対象)	2000-11-16
1	プロ野球 ファミスタ リターンズ	株式会社バンダイナムコエンターテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2015-10-08
2	プロ野球 ファミスタ 2011	株式会社バンダイナムコゲームス	ニンテンドー3DS	CERO A (全年齢対象)	2011-03-31
3	フロッガー3D	株式会社コナミデジタルエンタテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2011-09-22
4	ブレイブリーセカンド	株式会社スクウェア・エニックス	ニンテンドー3DS	CERO C (15才以上対象)	2015-04-23
...	...	...	...	...	...
21950	バイオショック インフィニット PlayStation Now版	テイクツー・インタラクティブ・ジャパン合同会社	プレイステーション Now	CERO D (17才以上対象)	2017-09-19
21951	バイオハザード リベレーションズ アンベールド エディション PlayStation Now版	株式会社カプコン	プレイステーション Now	CERO D (17才以上対象)	2016-09-20
21952	巫剣神威控 PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO C (15才以上対象)	2017-07-20
21953	ネバーエンディング ナイトメア PlayStation Now版	アクティブゲーミングメディア	プレイステーション	CERO D (17才以上対象)	2017-07-20

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
			Now		
21954	雷電IV OverKill PlayStation Now 版	モス	プレイステーション Now	CERO A (全年齢対象)	2017-02-21

21955 rows × 5 columns

また，取得したデータのうち目標クラスとなるゲームプラットフォームはどれくらいの種類があって，それぞれどれくらいの要素数を持つのか確認してみよう．

In [5]:
```
count = collections.Counter(games["プラットフォーム"])
count
```

Out[5]:
```
Counter({'ゲームボーイ': 1,
         'ニンテンドー3DS': 1993,
         'Xbox One': 645,
         'プレイステーション4': 1712,
         'プレイステーション・ポータブル': 3101,
         'Xbox 360': 738,
         'プレイステーション Vita': 1979,
         'プレイステーション3': 2307,
         'ドリームキャスト': 326,
         'セガサターン': 397,
         'Wii U': 817,
         'メガドライブ': 2,
         'ゲームボーイアドバンス': 233,
         'Wii': 998,
         'ニンテンドーDS': 2050,
         'ゲームアーカイブス': 945,
         'プレイステーション': 12,
         'プレイステーション2': 2707,
         'PCエンジン': 2,
         'Nintendo Switch': 315,
         'Microsoft Windows': 223,
         'macOS': 13,
         'ワンダースワン': 1,
         'ゲームギア': 5,
         'Xbox': 78,
         'ニンテンドーゲームキューブ': 128,
         '3DO': 54,
         'PC-FX': 3,
         'プレイステーション Now': 170})
```

今回は発売年月日のみによってある程度分類がうまくいくように，任天堂の家庭用ゲーム機のうち、据え置型のプラットフォーム（据置機）のみに絞ることとする．

In [6]:
```
new_classes = ["ニンテンドーゲームキューブ", "Wii", "Wii U", "Nintendo Switch"]
games = games[games['プラットフォーム'].isin(new_classes)]
collections.Counter(games["プラットフォーム"])
```

Out[6]:
```
Counter({'Wii U': 817,
         'Wii': 998,
         'Nintendo Switch': 315,
         'ニンテンドーゲームキューブ': 128})
```

データのベクトル化

データは揃ったが，このままタイトルや発行者などの文字列をモデルに直接入力することはできない．

そこで，今回特徴量にする情報群を，深層学習の章で用いたのと同じようにベクトル化していく．（本来決定木ではカテゴリ変数をそのまま特徴量としてもいいのだが，性能比較の意味も込めて同じOne-hotベクトルによるエンコーディングで学習を行うこととする）

目標変数（クラス）のベクトル化

各ゲームプラットフォームをカテゴリ変数と見做し，one-hotベクトル表現によってこれを表すことにする．

In [7]:
```
# ワンホットベクトルへのエンコードを行うインスタンスlabel_encoderの作成
label_encoder = LabelEncoder()

# プラットフォームが何か?を表すワンホットベクトルへの学習・変換を同時に行う関数.変換後はyに格納
y = label_encoder.fit_transform(np.array(games['プラットフォーム']).reshape(-1,1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:116: DataConversion
Warning: A column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

説明変数のベクトル化

公開年月日を，公開年/月/日というふうに３つのベクトルに分けて表現することにする．

In [8]:
```
years = []
months = []
dates = []
for day in games['公開年月日']:
  y_m_d = day.split("-")
  years.append(int(y_m_d[0]))
  months.append(int(y_m_d[1]))
  dates.append(int(y_m_d[2]))

years = np.array(years).reshape(-1,1)
months = np.array(months).reshape(-1,1)
dates = np.array(dates).reshape(-1,1)

# 各特徴量を横並びに結合して,1つのゲームパッケージにつき1つの結合された特徴量ベクトルが対応するよう
X = np.concatenate([years, months, dates], axis=1)
X.shape
```

Out[8]: (2258, 3)

3次元の特徴量を持つデータが11120件作成された．

データの次元削減

この後で可視化のプロセスを行いたいので，PCA（主成分分析）を利用して特徴量次元を2次元に削減する．

In [9]:
```
# 正規化
X = StandardScaler().fit_transform(X)
```

```
model_pca = PCA(n_components=2)
X = model_pca.fit_transform(X)

print(f"PCA1の寄与率: {model_pca.explained_variance_ratio_[0]}, PCA2の寄与率: {model_pca.ex
```

PCA1の寄与率: 0.37726529229711236, PCA2の寄与率: 0.3255300556663163

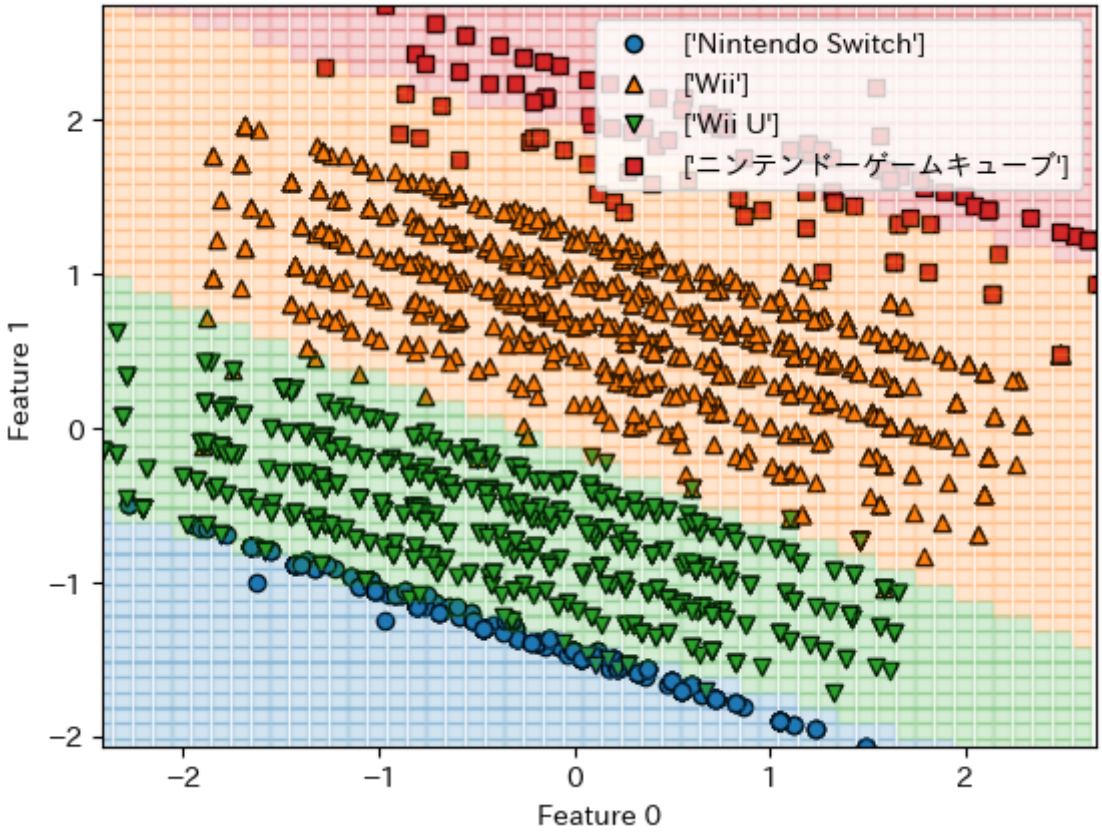
学習データとテストデータに分ける

In [10]:
```
# デフォルトでは学習データは75%,テストデータは25%になる
X_train, X_test, y_train, y_test = train_test_split(np.array(X), np.array(y), shuffle=True)
```

線形SVM

実際に分類を行ってみて，それを可視化しよう.

In [11]:
```
#モデルの作成・学習
lin_svm = LinearSVC().fit(X_train, y_train)

# 決定関数のy切片と傾き
b = lin_svm.intercept_
w = lin_svm.coef_

# 予測値
y_pred = lin_svm.predict(X_test)

# グラフの軸の範囲の設定
f0_min, f0_max = min(np.transpose(X)[0]), max(np.transpose(X)[0])
f1_min, f1_max = min(np.transpose(X)[1]), max(np.transpose(X)[1])

# グラフの点の種類,線の種類を指定
markers = ['o', '^', 'v', 's']
line_styles = ['-', '--', '-.', ':']
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red']

# プロット
fig, ax = plt.subplots()

for cls, marker, linestyle, color in zip(range(len(new_classes)), markers, line_styles, colors):
  #点のプロット
  x = X[y==cls]
  ax.scatter(x[:, 0], x[:, 1], c=[color], ec='k', marker=marker, label=label_encoder.inverse_transf


# 背景のプロット
for f0 in np.arange(f0_min, f0_max, 0.1):
  for f1 in np.arange(f1_min, f1_max, 0.1):
      conf = [b[cls] + w[cls, 0] * f0 + w[cls, 1] * f1 for cls in range(len(new_classes))]
      ax.scatter(f0, f1, c=colors[np.argmax(conf)], marker='s', s=40, alpha=0.2)

ax.set_xlim(f0_min, f0_max)
ax.set_ylim(f1_min, f1_max)
ax.set_xlabel("Feature 0")
ax.set_ylabel("Feature 1")
ax.legend()

plt.show()
```



上のグラフでは，背景の色の差で描かれる境界が「各クラスの決定関数の値を比べた時，最も値が大きいクラスが変わる部分」になる．

同じ発行者の同じ据置機のみが対象になっているということで，販売年月日からのみでも線形SVMでかなり綺麗にわけることができている．

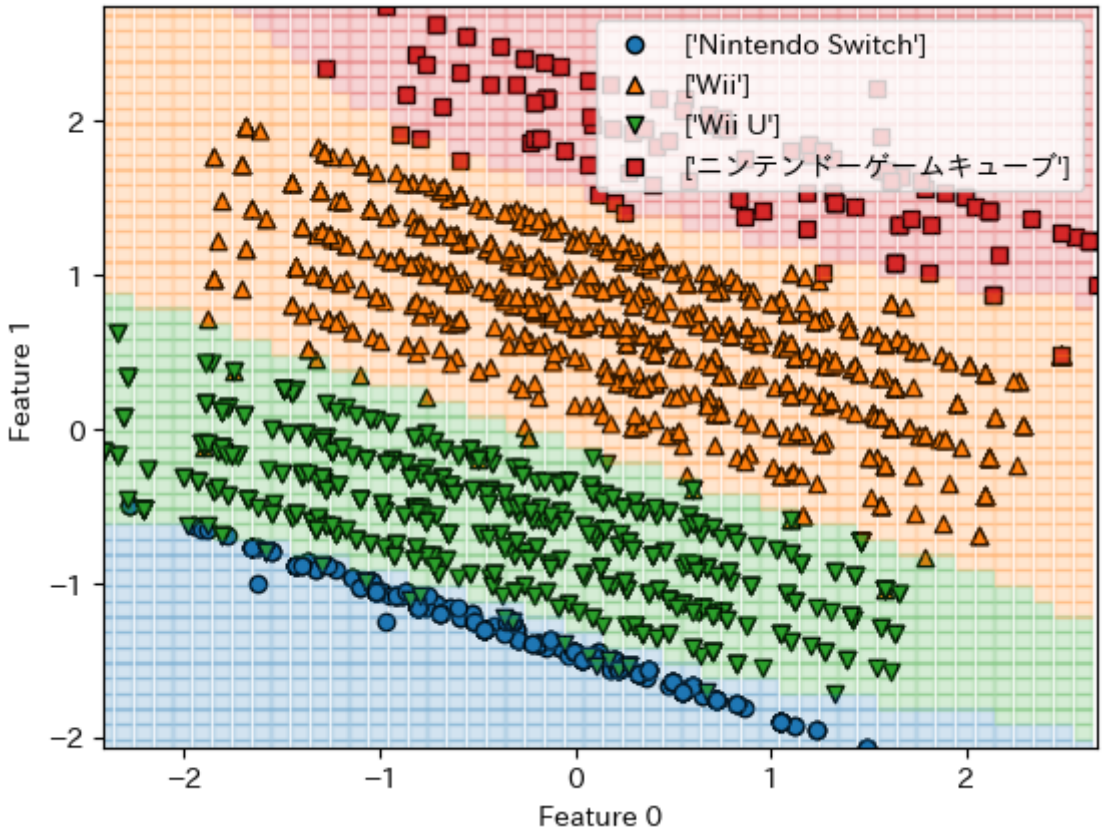
カーネルSVM

カーネルSVMを利用して同様の分類問題に挑戦してみよう． カーネルSVMではscikit-learnのSVCの引数kernelにさまざまな値を渡すことでさまざまなカーネル関数によるカーネルSVMを簡単に構築できる． 今回はRBFカーネルを利用してみよう．

In [12]:
```
from sklearn.svm import SVC
```

In [13]:
```
#モデルの作成・学習
rbf_svm = SVC(kernel='rbf').fit(X_train, y_train)

# 予測値
y_pred = rbf_svm.predict(X_test)

# グラフの軸の範囲の設定
f0_min, f0_max = min(np.transpose(X)[0]), max(np.transpose(X)[0])
f1_min, f1_max = min(np.transpose(X)[1]), max(np.transpose(X)[1])

# グラフの点の種類,線の種類を指定
markers = ['o', '^', 'v', 's']
line_styles = ['-', '--', '-.', ':']
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red']

# プロット
fig, ax = plt.subplots()

for cls, marker, linestyle, color in zip(range(len(new_classes)), markers, line_styles, colors):
```

```
  #点のプロット
  x = X[y==cls]
  ax.scatter(x[:, 0], x[:, 1], c=[color], ec='k', marker=marker, label=label_encoder.inverse_transf

# 背景のプロット
for f0 in np.arange(f0_min, f0_max, 0.1):
  for f1 in np.arange(f1_min, f1_max, 0.1):
      conf = rbf_svm.decision_function([[f0, f1]])
      ax.scatter(f0, f1, c=colors[np.argmax(conf)], marker='s', s=40, alpha=0.2)

ax.set_xlim(f0_min, f0_max)
ax.set_ylim(f1_min, f1_max)
ax.set_xlabel("Feature 0")
ax.set_ylabel("Feature 1")
ax.legend()

plt.show()
```



線形SVMと比べて，やや曲線的な境界になっていることがわかる．

3-6. 予測・判断 - 決定木（Decision Tree）、ランダムフォレスト、混同行列、Accuracy、Precision、Recall

「3-4. 深層学習の基礎と展望」で扱ったのと同様のデータセットに対して，今度は深層学習ではないさまざまな分類モデルを適応させていく． メディア芸術データベース・ラボ上のサービスを利用して取得したデータを用いて，簡単な分類を行う決定木・ランダムフォレストのサンプルコードを見ていこう．

データベース上からゲームパッケージを取得し，「発行者」「公開年月日」「対象年齢（CERO）」を特徴量化して，どのプラットフォームに該当するゲームなのかを当てる多クラス分類モデルを作っていく．

データの取得

メディア芸術データベース・ラボ上で提供されているSPARQLクエリサービスを利用して，ゲームパッケージのデータを取得していく．

以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる． 取得したCSVファイルを「ゲームパッケージ.csv」として保存し，このノートブック上にアップロードしよう．

※「3-4. ディープニューラルネットワーク（DNN）」，「3-4. 学習用データと学習済みモデル」，「3-6. サポートベクターマシン」，「3-7. 形態素解析」，「3-7. かな漢字変換」，「3-7. 表現学習」と同じデータセットを作成するため，もし同じものを持っている場合は以下の取得作業は不要なので，そちらをアップロードしよう．

```
PREFIX ma: <https://mediaarts-db.bunka.go.jp/data/property#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?発行者 ?プラットフォーム ?レーティング ?公開年月日
WHERE {
  ?アイテム
        schema:genre "ゲームパッケージ";
        schema:contentRating ?レーティング;
        schema:gamePlatform ?プラットフォーム;
        schema:publisher ?発行者;
        rdfs:label ?ラベル;
        ma:datePublished ?公開年月日.
}
```

取得したデータセットを，Python上で扱いやすい形にしていく． pandasというライブラリを用いて，csv形式のファイルの中身をDataFrameという型に変換し，扱っていく．

In [ ]:
```
import pandas as pd
import collections
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

In [ ]:
```
games = pd.read_csv("/content/ゲームパッケージ.csv")
games = games.drop_duplicates()
games = games.reset_index(drop=True)
games
```

Out[ ]:

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
0	Dance Dance Revolution GB2	コナミ株式会社	ゲームボーイ	CERO D (17才以上対象)	2000-11-16
1	プロ野球 ファミスタ リターンズ	株式会社バンダイナムコエンターテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2015-10-08
2	プロ野球 ファミスタ 2011	株式会社バンダイナムコゲームス	ニンテンドー3DS	CERO A (全年齢対象)	2011-03-31
3	フロッガー3D	株式会社コナミデジタルエンタテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2011-09-22
4	ブレイブリーセカンド	株式会社スクウェア・エニックス	ニンテンドー3DS	CERO C (15才以上対象)	2015-04-23
...	...	...	...	...	...
21976	バイオショック インフィニット PlayStation Now版	テイクツー・インタラクティブ・ジャパン合同会社	プレイステーション Now	CERO D (17才以上対象)	2017-09-19
21977	バイオハザード リベレーションズ アンベールド エディション PlayStation Now版	株式会社カプコン	プレイステーション Now	CERO D (17才以上対象)	2016-09-20
21978	巫剣神威控 PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO C (15才以上対象)	2017-07-20
21979	ネバーエンディング ナイトメア PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO D (17才以上対象)	2017-07-20
21980	雷電IV OverKill PlayStation Now版	モス	プレイステーション Now	CERO A (全年齢対象)	2017-02-21

21981 rows × 5 columns

データクレンジング

想定外の値を持つデータを除外する作業をする． 例えば， `games` の["公開年月日"]列には `yyyy-mm-dd` の形式で日付が入っていることが期待されるが，そうでない形式のものが含まれている可能性もある． 実際に見てみよう．

In [ ]:
```
for day in games['公開年月日']:
  if len(day) < 10:
    print(day)
```

2007
2000
2009
2002
2006
2006
2006
2001
2014
2002
1999
1998
2007
2003
2008
2001
2003
2013
2017
2001
2002
2001
2005
1998
1998
2006

こういったものを先に除外しておくと，後の計算が楽に正しく行えるようになる．

In [ ]:
```
games = games[games['公開年月日'].str.len() >= 10]
games
```

Out[ ]:

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
0	Dance Dance Revolution GB2	コナミ株式会社	ゲームボーイ	CERO D (17才以上対象)	2000-11-16
1	プロ野球 ファミスタ リターンズ	株式会社バンダイナムコエンターテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2015-10-08
2	プロ野球 ファミスタ 2011	株式会社バンダイナムコゲームス	ニンテンドー3DS	CERO A (全年齢対象)	2011-03-31
3	フロッガー3D	株式会社コナミデジタルエンタテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2011-09-22
4	ブレイブリーセカンド	株式会社スクウェア・エニックス	ニンテンドー3DS	CERO C (15才以上対象)	2015-04-23
...	...	...	...	...	...
21976	バイオショック インフィニット PlayStation Now版	テイクツー・インタラクティブ・ジャパン合同会社	プレイステーション Now	CERO D (17才以上対象)	2017-09-19

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
21977	バイオハザード リベレーションズ アンベールド エディション PlayStation Now版	株式会社カプコン	プレイステーション Now	CERO D (17才以上対象)	2016-09-20
21978	巫剣神威控 PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO C (15才以上対象)	2017-07-20
21979	ネバーエンディング ナイトメア PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO D (17才以上対象)	2017-07-20
21980	雷電IV OverKill PlayStation Now版	モス	プレイステーション Now	CERO A (全年齢対象)	2017-02-21

21955 rows × 5 columns

また，取得したデータのうち目標クラスとなるゲームプラットフォームはどれくらいの種類があって，それぞれどれくらいの要素数を持つのか確認してみよう．

In [ ]:
```
count = collections.Counter(games["プラットフォーム"])
count
```

Out[ ]:
```
Counter({'ゲームボーイ': 1,
         'ニンテンドー3DS': 1993,
         'Xbox One': 645,
         'プレイステーション4': 1712,
         'プレイステーション・ポータブル': 3101,
         'Xbox 360': 738,
         'プレイステーション Vita': 1979,
         'プレイステーション3': 2307,
         'ドリームキャスト': 326,
         'セガサターン': 397,
         'Wii U': 817,
         'メガドライブ': 2,
         'ゲームボーイアドバンス': 233,
         'Wii': 998,
         'ニンテンドーDS': 2050,
         'ゲームアーカイブス': 945,
         'プレイステーション': 12,
         'プレイステーション2': 2707,
         'PCエンジン': 2,
         'Nintendo Switch': 315,
         'Microsoft Windows': 223,
         'macOS': 13,
         'ワンダースワン': 1,
         'ゲームギア': 5,
         'Xbox': 78,
         'ニンテンドーゲームキューブ': 128,
         '3DO': 54,
         'PC-FX': 3,
         'プレイステーション Now': 170})
```

極端に少ない要素数のクラスを含めると，約30クラスもの分類問題になってしまう． 問題設定を簡単にするため， 要素数の多い（100個以上）家庭用ゲーム機の据置機のみに絞ることとす

る.

In [ ]:
```
new_classes = [key for key, value in count.items() if value >= 100]
new_classes = list(set(new_classes) & set(["プレイステーション", "プレイステーション2", "プレイステー
games = games[games['プラットフォーム'].isin(new_classes)]
collections.Counter(games["プラットフォーム"])
```

Out[ ]:
```
Counter({'Xbox One': 645,
         'プレイステーション4': 1712,
         'Xbox 360': 738,
         'プレイステーション3': 2307,
         'ドリームキャスト': 326,
         'Wii U': 817,
         'Wii': 998,
         'プレイステーション2': 2707,
         'Nintendo Switch': 315,
         'ニンテンドーゲームキューブ': 128})
```

データのベクトル化

データは揃ったが，このままタイトルや発行者などの文字列をモデルに直接入力することはできない.

そこで，今回特徴量にする情報群を，深層学習の章で用いたのと同じようにベクトル化していく．（本来決定木ではカテゴリ変数をそのまま特徴量としてもいいのだが，性能比較の意味も込めて同じOne-hotベクトルによるエンコーディングで学習を行うこととする）

目標変数（クラス）のベクトル化

各ゲームプラットフォームをカテゴリ変数と見做し，one-hotベクトル表現によってこれを表すことにする.

In [ ]:
```
# ワンホットベクトルへのエンコードを行うインスタンスonehot_encoderの作成
onehot_encoder = OneHotEncoder(sparse = False,dtype = int)

# プラットフォームが何か?を表すワンホットベクトルへの学習・変換を同時に行う関数.変換後はyに格納
y = onehot_encoder.fit_transform(np.array(games['プラットフォーム']).reshape(-1,1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarni
ng: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse
_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

説明変数のベクトル化

発行者のベクトル化

各会社をカテゴリ変数と見做し，one-hotベクトル表現によってこれを表すことにする.

In [ ]:
```
# 出版元がどこか?を表すワンホットベクトルへの学習・変換を同時に行う関数.変換後はcreatorsに格納
creators = onehot_encoder.fit_transform(np.array(games['発行者']).reshape(-1,1))
creators.shape
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarni
ng: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse
_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

Out[ ]: (10693, 756)

レートのベクトル化

各レートをカテゴリ変数と見做し，one-hotベクトル表現によってこれを表すことにする.

In [ ]:
```
# 対象年齢がどれか?を表すワンホットベクトルへの学習・変換を同時に行う関数.変換後はratesに格納
rates = onehot_encoder.fit_transform(np.array(games['レーティング']).reshape(-1,1))
rates.shape
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarni
ng: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse
_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

Out[ ]: (10693, 14)

公開年月日のベクトル化

公開年/月/日というふうに３つのベクトルに分けて表現することにする.

In [ ]:
```
years = []
months = []
dates = []
for day in games['公開年月日']:
  y_m_d = day.split("-")
  years.append(int(y_m_d[0]))
  months.append(int(y_m_d[1]))
  dates.append(int(y_m_d[2]))

years = np.array(years).reshape(-1,1)
months = np.array(months).reshape(-1,1)
dates = np.array(dates).reshape(-1,1)

years.shape
```

Out[ ]: (10693, 1)

説明変数（特徴量）ベクトルの作成

In [ ]:
```
# 各特徴量を横並びに結合して,1つのゲームパッケージにつき1つの結合された特徴量ベクトルが対応するよう
X = np.concatenate([creators, rates, years, months, dates], axis=1)

print(f"{X.shape[1]}次元の特徴量を持つデータが{X.shape[0]}件作成された.")
```

773次元の特徴量を持つデータが10693件作成された.

学習データとテストデータに分ける

In [ ]:
```
# デフォルトでは学習データは75%,テストデータは25%になる
X_train, X_test, y_train, y_test = train_test_split(np.array(X), np.array(y), shuffle=True)
```

決定木モデルの構築

上記で用意したデータセットを利用して，簡単な多クラス分類決定木モデルを作っていく.

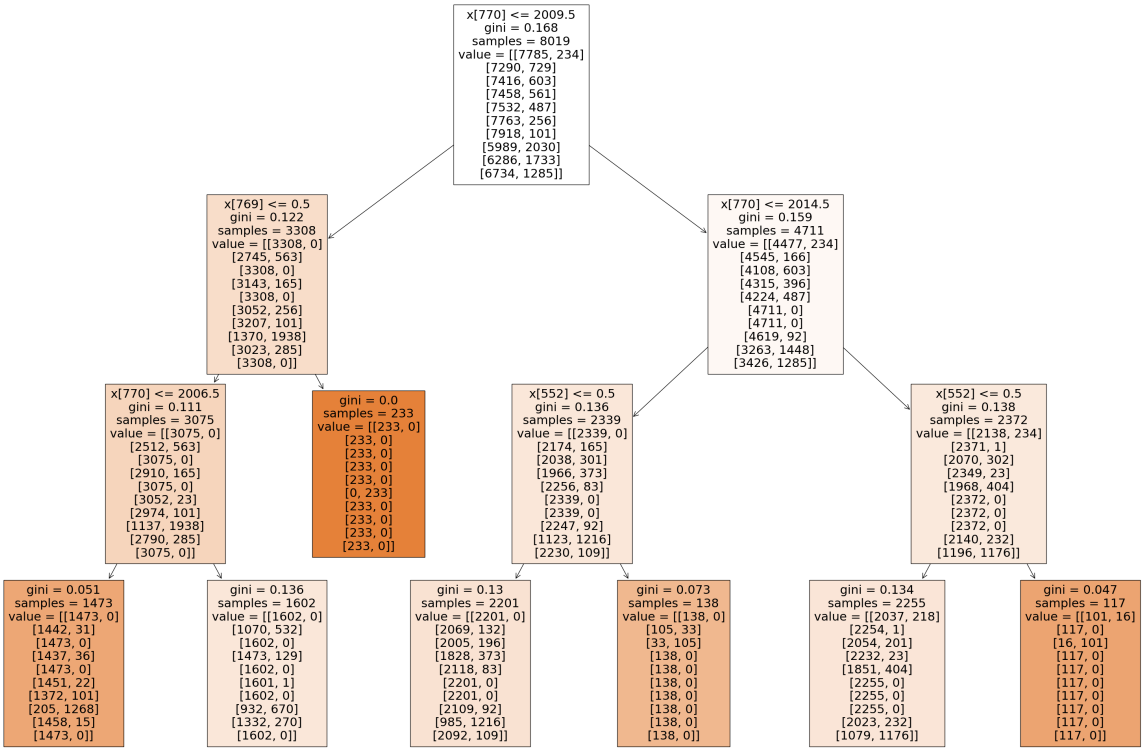
In [ ]:
```
# 必要なモジュールの読み込み
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import sklearn.tree
```

決定木がどんなふうに分類を行っているのか，簡易化のために深さを3までに制限した決定木を作成し，プロットしてみよう．

In [ ]:
```
tree = DecisionTreeClassifier(max_depth=3) #デフォルト値(None)では最大まで展開される
tree.fit(X_train, y_train)
print("正解率: ", tree.score(X_test, y_test))
# 決定木を可視化
plt.figure(figsize=(40, 25))
sklearn.tree.plot_tree(tree, # モデルの名前
                       class_names=["CERO A","CERO B","CERO C","CERO D","CERO Z"], # クラス名
                       filled=True # 色付きで表示
                       );
```

正解率: 0.5003739715781601



見てみると，各要素がある閾値以下かどうかで場合分けが進んでいる様子がわかる． このような処理を何度も何度も繰り返すことで決定木は分類性能を向上させていく． 場合わけ処理の多さ=木の深さが深い方が訓練データでの正解率は上がるが，もちろん深いほどいいというわけではなく過学習の原因にもなりうるので，適切に深さを指定することも大切である．

max_depthを指定しなかった場合は正解率は以下のようになる．

In [ ]:
```
tree = DecisionTreeClassifier() #デフォルト(None)では最大まで展開される
tree.fit(X_train, y_train)
print("正解率: ", tree.score(X_test, y_test))
print("木の深さ: ", tree.get_depth())
```

正解率: 0.6387434554973822
木の深さ: 45

3-6. 予測・判断 - ランダムフォレスト

上記で用意したデータセットを利用して，決定木と同様に簡単な多クラス分類ランダムフォレストを作っていく．

ランダムフォレストとは複数の決定木（いわゆる"森"）を組合せることで推定精度を向上させる手法だが，Pythonでは決定木モデルを何個も作って学習していく必要はなく，こちらも`RandomForestClassifier`というライブラリがscikit-learnで用意されている．そのため，以下のように簡単に構築することができる．

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier() # デフォルトで100個の木の組み合わせ,最大まで展開される
random_forest.fit(X_train, y_train)

print("正解率: ", random_forest.score(X_test, y_test))
```

正解率: 0.6537023186237846

3-6. 予測・判断 - 特徴量の効果的な選択

ランダムフォレストでは，各決定木モデルは全ての特徴量は使わずに学習しており，それらを組み合わせてモデル全体を学習していくため，どのモデルに重きを置くか=どの特徴量に重きを置くかということで重要な特徴量を選ぶということを行なっている．

構築したモデルが各特徴量についてどのように重要度を計算したのか見てみると，次のようになる．

In [ ]:
```
random_forest.feature_importances_
```

Out[ ]:
```
array([0.00000000e+00, 3.51990011e-05, 0.00000000e+00, 2.90741005e-04,
       1.70935270e-03, 2.67402190e-04, 1.06406329e-04, 1.20711627e-04,
       6.67389719e-04, 1.76063952e-04, 1.03937260e-03, 1.06439173e-04,
       6.94695325e-04, 3.93086042e-04, 0.00000000e+00, 4.63970805e-05,
       9.08230215e-05, 1.27164410e-04, 5.36967571e-05, 3.23234752e-05,
       2.70374343e-04, 0.00000000e+00, 5.85307278e-05, 3.79244306e-05,
       7.01888752e-05, 5.96210056e-05, 1.14912267e-03, 9.40738215e-05,
       1.06940309e-04, 0.00000000e+00, 6.16312553e-05, 1.75973773e-04,
       0.00000000e+00, 0.00000000e+00, 4.87358093e-05, 1.05506954e-04,
       4.91737107e-06, 2.58945966e-05, 2.28811936e-05, 1.50519604e-04,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.31328222e-04,
       5.41322765e-05, 5.08557267e-05, 1.38027107e-04, 0.00000000e+00,
       6.94950085e-04, 2.33830304e-04, 9.55707679e-05, 1.24542233e-04,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.70093723e-04,
       2.00773897e-04, 6.06284528e-05, 1.30083400e-04, 8.74112883e-05,
       5.72612128e-05, 0.00000000e+00, 2.20409573e-04, 1.77363782e-04,
       2.21165431e-04, 1.45568346e-04, 1.21430170e-04, 5.48196842e-05,
       2.01452572e-04, 8.14337682e-04, 5.08759541e-04, 8.55717623e-05,
       1.17471144e-04, 0.00000000e+00, 0.00000000e+00, 1.46890116e-04,
       1.66378357e-04, 0.00000000e+00, 4.55888290e-04, 6.06880827e-04,
       4.82388875e-05, 4.75112559e-05, 4.25376083e-05, 1.32578226e-04,
       7.40366582e-10, 0.00000000e+00, 0.00000000e+00, 9.02086458e-05,
       1.36096807e-04, 0.00000000e+00, 1.16626024e-04, 0.00000000e+00,
       1.26015277e-04, 6.25431706e-05, 2.81023130e-04, 6.72675818e-05,
       3.20706050e-05, 0.00000000e+00, 5.51417762e-05, 1.14070340e-04,
       9.30973508e-05, 5.17587541e-05, 6.07580262e-05, 1.13945833e-04,
       3.01478635e-04, 7.83473550e-06, 1.32087818e-04, 7.65837560e-05,
```

2.98787214e-04, 1.64245299e-04, 1.28290573e-04, 0.00000000e+00,
5.13730657e-05, 1.18101390e-04, 0.00000000e+00, 7.21446249e-05,
1.09407513e-04, 5.95117605e-05, 5.10699759e-05, 0.00000000e+00,
1.82975729e-04, 2.42003001e-05, 1.35033483e-04, 1.09681186e-05,
1.45744391e-04, 4.09044140e-05, 4.85535953e-05, 4.26403118e-05,
0.00000000e+00, 4.19813625e-05, 5.97556960e-05, 1.44944377e-04,
1.40183071e-04, 0.00000000e+00, 1.56566172e-04, 9.65726978e-05,
7.34898111e-05, 3.66714508e-05, 2.39845825e-05, 6.11643486e-05,
1.35803915e-04, 0.00000000e+00, 1.26753516e-04, 7.59815769e-04,
1.21733116e-04, 1.80304228e-04, 1.18744886e-04, 0.00000000e+00,
4.59028844e-04, 3.01750002e-04, 8.49919335e-03, 1.16658889e-04,
4.27315160e-05, 0.00000000e+00, 0.00000000e+00, 1.30975897e-04,
1.87018696e-04, 1.66555653e-04, 2.76333169e-04, 1.00151863e-04,
1.50914150e-05, 1.48528111e-05, 1.52012385e-03, 1.51520564e-05,
2.17357922e-05, 0.00000000e+00, 0.00000000e+00, 1.56712226e-04,
0.00000000e+00, 2.71094647e-04, 1.28514751e-04, 1.37262852e-04,
1.49106656e-04, 1.00910481e-04, 0.00000000e+00, 1.36529943e-04,
1.15037354e-04, 3.77676112e-05, 0.00000000e+00, 1.63621794e-04,
1.22942147e-04, 1.13567881e-04, 4.15844150e-05, 0.00000000e+00,
8.70980632e-05, 3.56217178e-05, 2.63354650e-04, 2.85284019e-05,
1.25705861e-04, 3.46549723e-05, 7.17367442e-05, 0.00000000e+00,
5.92670501e-05, 2.02811260e-04, 1.31649922e-04, 1.31364667e-05,
3.21336881e-05, 5.48127424e-05, 1.87872453e-04, 2.43057711e-04,
3.10629740e-04, 7.96429717e-05, 6.64745041e-05, 1.68957304e-04,
5.63285963e-05, 3.58320138e-05, 2.26134816e-04, 0.00000000e+00,
2.94228604e-05, 0.00000000e+00, 1.22039571e-04, 4.64985086e-04,
3.87784327e-04, 8.54621672e-05, 0.00000000e+00, 2.14231811e-04,
2.45880609e-05, 5.75090246e-05, 6.68670002e-05, 1.33110820e-04,
3.34797727e-04, 9.17678230e-05, 5.27771765e-05, 0.00000000e+00,
1.41066288e-04, 1.48520023e-04, 8.83771041e-05, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.88078106e-05,
1.04432551e-04, 4.27188616e-05, 3.25163981e-05, 5.64779269e-05,
9.89031846e-05, 5.14394690e-05, 0.00000000e+00, 7.96081633e-05,
1.13586534e-04, 4.24161551e-05, 2.64883881e-03, 1.22750841e-05,
8.88934834e-05, 4.83743721e-05, 1.58929534e-04, 1.70147246e-04,
0.00000000e+00, 9.31649537e-04, 9.26221071e-05, 3.07548599e-04,
5.19763084e-05, 1.20267246e-04, 9.54201370e-05, 1.41473174e-04,
2.66169354e-04, 0.00000000e+00, 7.05906924e-05, 1.57009101e-04,
1.59977920e-04, 2.42347533e-04, 4.39016036e-04, 3.60624633e-04,
0.00000000e+00, 1.45250976e-04, 4.08544564e-05, 4.01078909e-05,
1.58639464e-04, 1.32628132e-04, 9.89175732e-05, 1.29083496e-04,
1.01808478e-04, 0.00000000e+00, 1.26925698e-04, 3.30357331e-04,
6.85476336e-05, 1.07456512e-04, 1.29240938e-04, 1.52119134e-04,
3.10741442e-04, 5.04271125e-05, 1.76886718e-04, 1.43548198e-04,
6.93309883e-05, 9.19815153e-05, 1.08354282e-04, 8.95422230e-05,
1.57810701e-04, 2.80268905e-04, 2.91698409e-05, 6.65739151e-05,
1.30775561e-04, 1.40407754e-04, 1.57843224e-04, 1.18335152e-04,
6.82991019e-05, 6.25988420e-04, 3.49929913e-05, 1.30078405e-04,
5.13689890e-03, 5.16132328e-06, 2.61142809e-05, 1.18785055e-03,
5.15816522e-05, 1.50119110e-03, 2.07245203e-04, 9.61384145e-05,
2.06378253e-05, 5.34751732e-05, 0.00000000e+00, 1.75817354e-04,
0.00000000e+00, 3.70716836e-05, 2.06095961e-04, 1.04795375e-04,
0.00000000e+00, 1.07320995e-04, 6.44613698e-06, 7.22946138e-04,
7.89658815e-06, 1.07882574e-05, 2.43789878e-05, 1.01653362e-05,
3.81741687e-05, 4.10405098e-03, 1.30422498e-05, 1.77125659e-05,
6.71877592e-05, 3.64125155e-05, 1.56059721e-04, 2.75999050e-04,
1.23387781e-03, 7.43130846e-05, 3.47327657e-06, 4.79247775e-05,
9.71066691e-04, 9.22421592e-05, 4.72166649e-05, 4.32579885e-05,

6.44400590e-04, 1.88738521e-04, 0.00000000e+00, 4.21245840e-04,
7.16999747e-06, 1.34211164e-04, 1.37195771e-04, 1.09190199e-04,
2.61119666e-04, 8.78057943e-03, 0.00000000e+00, 7.14401806e-04,
1.81320340e-05, 8.37816362e-05, 3.54235007e-04, 1.58875892e-04,
9.94625307e-06, 4.00261511e-03, 2.05762538e-05, 9.25143771e-05,
5.63310161e-05, 8.27563834e-05, 4.03026161e-04, 3.28520409e-04,
5.30582465e-04, 4.32031767e-05, 0.00000000e+00, 1.23044111e-03,
6.89472014e-06, 1.87195604e-06, 8.29081153e-05, 4.73424376e-05,
0.00000000e+00, 1.25853676e-04, 9.72520244e-06, 9.11815990e-04,
2.49918621e-04, 2.19370283e-04, 2.65171222e-05, 1.41319732e-04,
1.44760347e-05, 5.43751386e-05, 2.57527019e-03, 1.95886998e-03,
6.46778777e-05, 1.23064951e-04, 1.31493209e-04, 4.30530570e-04,
9.28620204e-04, 2.34732268e-03, 4.57896645e-05, 2.29364631e-03,
1.58076971e-04, 3.13547451e-04, 2.26676585e-04, 4.63766340e-04,
4.68129212e-05, 5.58180157e-04, 1.28461031e-05, 2.50224316e-03,
7.30589730e-04, 1.99746518e-04, 3.39076270e-04, 1.01090869e-04,
4.95518480e-05, 2.40780564e-05, 3.87609567e-05, 2.99988654e-05,
1.10067873e-04, 1.03672385e-04, 1.65824807e-05, 3.72870181e-03,
5.16414910e-05, 1.50846076e-04, 4.28066574e-06, 1.42853705e-04,
2.39730955e-07, 8.76047341e-06, 8.19323883e-04, 7.41556323e-04,
3.48051773e-05, 1.05417984e-04, 0.00000000e+00, 2.67669831e-03,
1.02310704e-03, 3.33261042e-03, 0.00000000e+00, 3.18766164e-06,
5.63784341e-05, 6.49597916e-04, 2.27568370e-04, 0.00000000e+00,
4.76215521e-03, 3.04370015e-05, 1.74517450e-05, 1.89736432e-03,
4.62868127e-05, 1.41309494e-04, 0.00000000e+00, 2.17455306e-05,
7.55169649e-05, 1.79762656e-05, 1.19428797e-04, 2.68617736e-04,
1.17598381e-05, 7.54570697e-05, 0.00000000e+00, 1.09141985e-05,
1.74966536e-05, 3.33080777e-05, 3.93834379e-05, 2.96869187e-03,
1.11931127e-03, 4.06818925e-05, 2.26426602e-04, 5.45417323e-05,
1.37488816e-04, 7.09624159e-05, 6.54455646e-04, 0.00000000e+00,
1.44560507e-05, 4.62056480e-05, 9.98321209e-05, 1.01401945e-04,
4.66060936e-05, 2.34305367e-04, 0.00000000e+00, 1.15659510e-05,
1.59033330e-05, 1.96004533e-04, 0.00000000e+00, 0.00000000e+00,
6.32041201e-04, 1.60998809e-05, 2.78618459e-05, 0.00000000e+00,
1.84923895e-04, 0.00000000e+00, 1.61195670e-05, 5.17708677e-03,
7.83721188e-04, 4.12012357e-03, 2.19542934e-03, 2.10053367e-04,
2.50518861e-04, 7.86461350e-05, 1.65033366e-04, 0.00000000e+00,
2.42118172e-05, 1.72527841e-05, 1.30283363e-05, 1.68419189e-04,
1.34764876e-04, 4.14669953e-06, 2.47672083e-05, 1.23778203e-04,
1.57248562e-04, 4.39215765e-05, 2.04687352e-05, 1.67642730e-03,
1.13672041e-04, 8.05090931e-04, 8.38382764e-05, 3.48089744e-05,
9.60426185e-05, 1.66275743e-04, 8.47904989e-04, 3.58902897e-04,
1.43771608e-04, 0.00000000e+00, 2.20660866e-04, 6.27729136e-04,
4.53791293e-04, 1.33201980e-05, 2.34977913e-04, 9.26795476e-04,
1.18413233e-05, 3.36374472e-05, 8.41077865e-04, 2.04043066e-04,
3.23978956e-04, 1.65121188e-04, 5.80281182e-04, 1.25110891e-04,
7.87530256e-06, 3.38979986e-04, 1.97212281e-04, 3.50021370e-04,
7.01317611e-04, 9.22337865e-05, 6.43664250e-03, 6.79943838e-05,
1.59361350e-04, 2.85476620e-04, 3.11155627e-05, 1.07259059e-08,
1.30023358e-05, 1.61415306e-03, 0.00000000e+00, 5.03045048e-05,
1.05398649e-04, 5.50537397e-04, 2.61969694e-05, 3.55120712e-04,
1.94900996e-05, 0.00000000e+00, 1.72611034e-03, 1.17474106e-04,
1.02829290e-03, 4.89706844e-04, 2.79586653e-04, 6.82389349e-05,
2.79511076e-02, 0.00000000e+00, 4.76366985e-04, 6.58368753e-05,
1.33459274e-03, 5.09979175e-06, 3.92521937e-05, 0.00000000e+00,
5.02518462e-05, 1.53133092e-04, 1.89051695e-05, 1.65712284e-05,
2.18605813e-04, 4.52562006e-04, 8.57105827e-04, 7.21414842e-03,
1.49069809e-03, 8.61096898e-09, 0.00000000e+00, 1.18113528e-04,

1.11710395e-04, 4.99401106e-05, 2.45866958e-04, 4.57147746e-04,
8.72358647e-04, 4.76704461e-04, 3.71791554e-04, 5.07292661e-04,
1.87757287e-05, 8.03715282e-05, 3.50874103e-05, 2.39992931e-04,
6.78218521e-04, 4.10177886e-04, 1.33326953e-05, 3.09048543e-03,
2.96282371e-04, 5.04321199e-05, 0.00000000e+00, 1.22876680e-04,
9.39999942e-04, 4.13083731e-05, 1.83703374e-03, 1.37694875e-04,
3.79756763e-04, 2.58481766e-04, 1.72047731e-03, 7.06649431e-04,
8.43768487e-04, 2.27015064e-04, 5.36707716e-04, 3.84468448e-05,
1.59603404e-04, 7.77943314e-05, 1.22253609e-03, 1.55546215e-04,
1.93348084e-04, 1.11763395e-04, 9.54750888e-05, 1.94725446e-04,
2.65834835e-04, 1.10609374e-03, 1.38228307e-03, 1.72337787e-04,
1.93205980e-04, 8.36611686e-05, 4.83354486e-05, 1.20308332e-04,
6.16884427e-05, 1.09330025e-04, 6.03648155e-05, 5.82174639e-04,
5.14070365e-04, 1.81559366e-04, 9.23954371e-05, 9.79718764e-05,
2.48256855e-04, 4.71699413e-05, 0.00000000e+00, 2.27818419e-03,
7.58540531e-03, 8.31056482e-05, 1.30940593e-03, 3.72017612e-05,
1.35086842e-03, 1.56231051e-04, 6.73817939e-05, 9.12727125e-05,
3.85446073e-05, 5.72002835e-05, 1.84958099e-03, 0.00000000e+00,
6.52301129e-05, 3.98925198e-05, 6.78777887e-03, 4.21974158e-05,
1.60166969e-03, 5.34274523e-03, 7.25729138e-03, 7.34790771e-05,
1.58545349e-03, 1.18600319e-03, 1.33179337e-04, 2.44700824e-04,
9.93655847e-05, 1.33686813e-04, 7.99659682e-04, 2.81461316e-04,
1.43558872e-04, 9.94548346e-05, 6.34239529e-03, 3.27402749e-04,
7.95455132e-05, 2.44086101e-03, 3.91153029e-03, 2.49986068e-04,
1.50815775e-04, 8.25929015e-03, 1.66247239e-03, 1.87906756e-03,
1.02824659e-04, 1.11567785e-02, 4.22566384e-05, 8.69580794e-04,
1.10034597e-04, 1.21509618e-03, 3.52160291e-04, 9.58243080e-05,
1.18979508e-04, 3.93114721e-03, 5.65064945e-05, 2.63807391e-04,
5.64050329e-05, 1.64186718e-04, 1.72434682e-04, 5.07230514e-05,
1.93407928e-04, 5.33165097e-05, 5.46698299e-04, 6.78317519e-04,
4.02914386e-05, 1.04381633e-02, 5.78412947e-04, 3.35500876e-03,
1.98135590e-04, 3.25203442e-03, 3.06533566e-03, 8.52340980e-03,
1.01304026e-03, 2.57781945e-04, 8.20465273e-05, 3.91242535e-05,
0.00000000e+00, 1.43528327e-04, 1.01193227e-04, 0.00000000e+00,
3.36780298e-05, 1.70670175e-03, 9.59156744e-04, 6.80051403e-04,
1.13919780e-04, 0.00000000e+00, 6.52441818e-05, 8.96092384e-04,
3.31425422e-05, 6.87409155e-04, 3.85416945e-05, 2.26050256e-05,
5.12776689e-04, 1.67678591e-04, 2.02731288e-03, 2.71449379e-04,
1.12583868e-04, 4.24789041e-04, 9.88317839e-05, 4.32835167e-04,
0.00000000e+00, 7.65367882e-04, 1.33695345e-04, 5.94791696e-04,
1.03598953e-04, 2.26853755e-04, 8.94882453e-05, 5.50682811e-05,
1.15689876e-05, 1.18434930e-03, 3.00596170e-04, 8.03478167e-06,
6.29655818e-04, 2.00540386e-03, 7.71705404e-04, 2.09921884e-04,
9.78878091e-04, 1.86677583e-03, 1.43970209e-03, 1.68591353e-04,
0.00000000e+00, 1.28322037e-05, 0.00000000e+00, 8.41867006e-04,
1.33000673e-04, 6.82716798e-04, 1.65915398e-05, 2.04073946e-05,
2.15286783e-03, 2.04063052e-03, 8.55439312e-04, 1.99875205e-02,
1.00832547e-02, 8.49333883e-03, 9.29775154e-03, 7.49593129e-03,
8.94395208e-03, 3.53831826e-04, 1.31991665e-04, 6.24268674e-05,
1.32689775e-04, 2.75839203e-02, 3.08384882e-01, 9.28851378e-02,
1.25297750e-01])

いくつかの値が0になっている．つまり，この値はランダムフォレストの計算に関与しない無意味なベクトルであったことがわかる．

特徴量はむやみに多ければ良いというわけではないので，適切に選択する必要がある．

3-6. 予測・判断 - 混同行列、Accuracy、Precision、Recall

In [ ]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

ここまでに作成してきた決定木・ランダムフォレストについて，モデルの評価を行ってみよう．

以下では分類モデルの評価を行う指標である混同行列・Accuracy・Precision・Recallを計算する手法について詳しく見ていく．

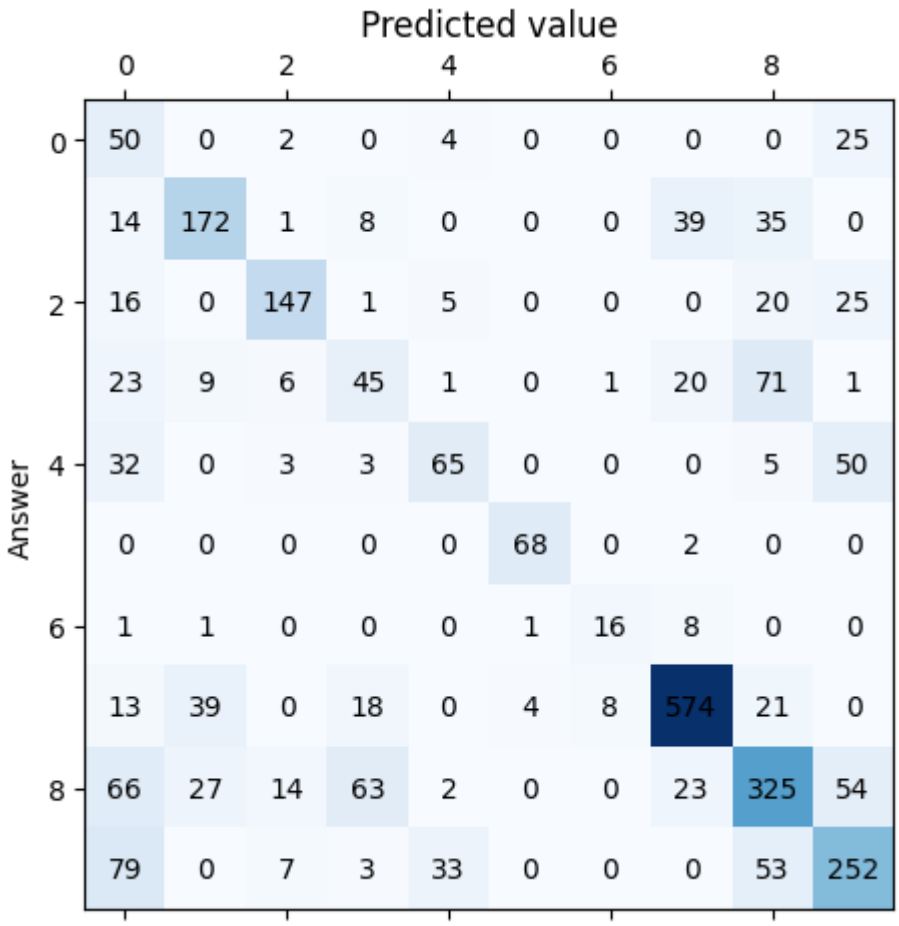
混同行列

縦軸・横軸に正解ラベル，予測ラベルのいずれかをとった行列で，縦軸に正解ラベルを取った場合であればi行j列目のxという値は，「決定木がクラス[j]と予測したもののうち，本当はラベル[i]のものがx個あった」ということを意味する． つまり，混同行列の左上から右下にかけての斜めのマスの値が大きく，それ以外の値が小さいモデルは良いモデルであると言える．

実際に混同行列を作成して比較してみよう．

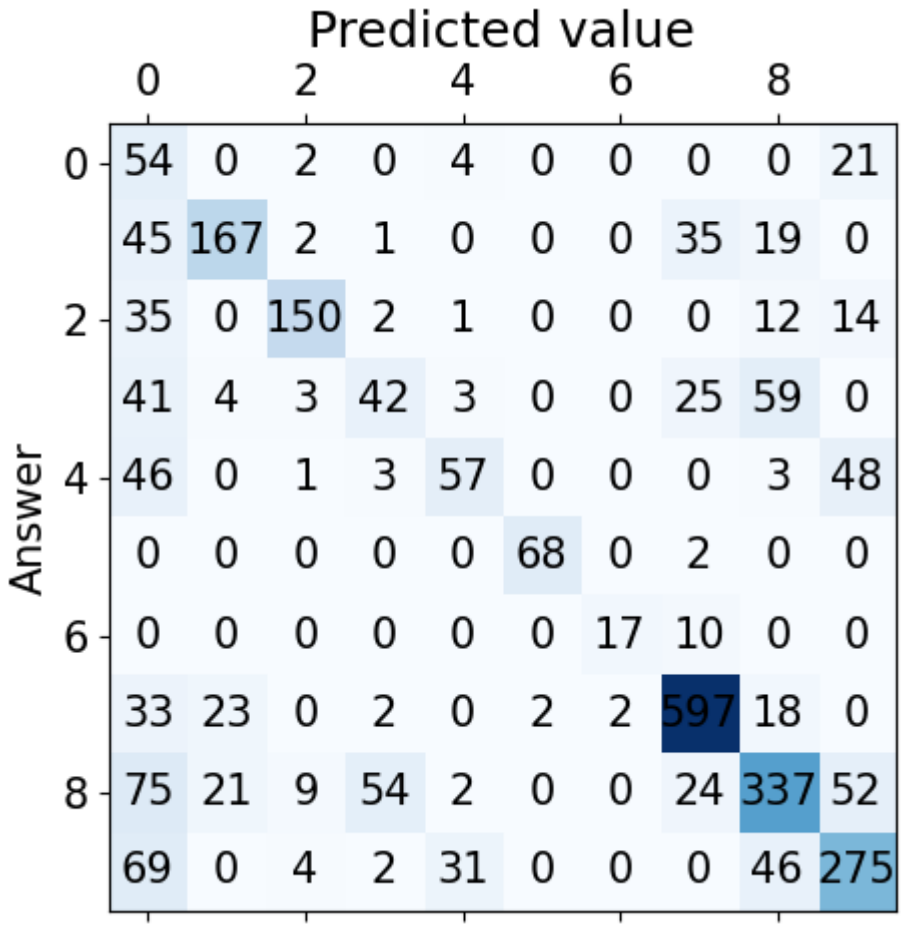
In [ ]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

In [ ]:
```
# 決定木
# 予測値を作成
y_pred_tree = tree.predict(X_test)

# 混同行列を作成
conf_mat_tree = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_pred_tree, axis=1)


# プロット
fig, ax = plt.subplots(figsize=(5, 5))
ax.matshow(conf_mat_tree, cmap=plt.cm.Blues)
for i in range(conf_mat_tree.shape[0]):
    for j in range(conf_mat_tree.shape[1]):
        ax.text(x=j, y=i, s=conf_mat_tree[i, j], va='center', ha='center')
plt.title('Predicted value')
plt.ylabel('Answer')
plt.rcParams["font.size"] = 15
plt.tight_layout()
plt.show()
```



In [ ]:

```
# ランダムフォレスト
# 予測値を作成
y_pred_forest = random_forest.predict(X_test)

# 混同行列を作成
conf_mat_forest = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_pred_forest, axis

# プロット
fig, ax = plt.subplots(figsize=(5, 5))
ax.matshow(conf_mat_forest, cmap=plt.cm.Blues)
for i in range(conf_mat_forest.shape[0]):
    for j in range(conf_mat_forest.shape[1]):
        ax.text(x=j, y=i, s=conf_mat_forest[i, j], va='center', ha='center')
plt.title('Predicted value')
plt.ylabel('Answer')
plt.rcParams["font.size"] = 15
plt.tight_layout()
plt.show()
```



正解率（Accuracy）

正解率とは「全体に対する正解の割合」であり，全てのテストデータのうち正解できた割合を指す．

正解率についての計算は一意に定めることができ，scikit-learnの `sklearn.metrics` モジュールを利用することで以下のように簡単に計算することができる．

In [ ]:
```
print("決定木")
print(f"正解率(Accuracy): {accuracy_score(y_test, y_pred_tree)}")
print()
print("ランダムフォレスト")
print(f"正解率(Accuracy): {accuracy_score(y_test, y_pred_forest)}")
```

```
決定木
正解率(Accuracy): 0.6387434554973822

ランダムフォレスト
正解率(Accuracy): 0.6537023186237846
```

適合率（Precision）

適合率とは「陽性と判定されたうち真の値が陽性である割合」であり，「どのクラスを陽性として扱うか」，「どのクラスを陰性として扱うか」によって値が変わってくる関係で計算の手法がいくつかある．

例えば

「各クラスごとに，自身を陽性，それ以外を全て陰性として（One vs Rest/OvR）適合率・再現率を計算してから平均をとる（マクロ平均）」のか，

「各クラスについてのTP/FP/FNの数を計算してからそれらの値を使ってモデル全体の適合率/再現率を計算する（マイクロ平均）」のかなどから手法を選ぶ必要がある．

マクロ平均で計算する場合は以下のように計算することができる．

In [ ]:
```
print("決定木")
print(f"適合率(Precision): {precision_score(y_test, y_pred_tree, average='macro')}")
print()
print("ランダムフォレスト")
print(f"適合率(Precision): {precision_score(y_test, y_pred_forest, average='macro')}")
```

```
決定木
適合率(Precision): 0.6688753684381503

ランダムフォレスト
適合率(Precision): 0.7416055157502504
```

マイクロ平均を利用する場合は，引数の値を変えることで以下のように計算することができる．

In [ ]:
```
print("決定木")
print(f"適合率(Precision): {precision_score(y_test, y_pred_tree, average='micro')}")
print()
print("ランダムフォレスト")
print(f"適合率(Precision): {precision_score(y_test, y_pred_forest, average='micro')}")
```

```
決定木
適合率(Precision): 0.6962902568283734

ランダムフォレスト
適合率(Precision): 0.7502145922746781
```

再現率（Recall）

再現率とは「真の値が陽性のうち，陽性と判定されたものの割合」である．

再現率の値も以下のように適合率と同様に計算することができる．

In [ ]:
```
# マクロ平均
print("決定木")
print(f"再現率(Recall): {recall_score(y_test, y_pred_tree, average='macro')}")
print()
print("ランダムフォレスト")
print(f"適合率(Precision): {recall_score(y_test, y_pred_forest, average='macro')}")
```

```
決定木
再現率(Recall): 0.6103406049025308

ランダムフォレスト
適合率(Precision): 0.6102961783122959
```

In [ ]:
```
# マイクロ平均
print("決定木")
print(f"再現率(Recall): {recall_score(y_test, y_pred_tree, average='micro')}")
print()
print("ランダムフォレスト")
print(f"適合率(Precision): {recall_score(y_test, y_pred_forest, average='micro')}")
```

決定木
再現率(Recall): 0.6387434554973822

ランダムフォレスト
適合率(Precision): 0.6537023186237846

3-7. 言語・知識 - かな漢字変換

いくつかの文章から単語を分割して取得したい時，同じ意味を持つ単語でも表記が漢字なのかひらがななのかによって別単語と認識してしまうと，想定と違う結果が得られてしまったりする． そのため，全ての漢字表記をひらがなに変換するなどの処理が行われることがある．実用上ではこれらの処理をどのようにして行っていくのか，ゲームパッケージのタイトルを利用して具体的にみてみよう．

データの取得・整形

メディア芸術データベース・ラボ上で提供されているSPARQLクエリサービスを利用して，ゲームパッケージのデータを取得していく．

以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる． 取得したCSVファイルを「ゲームパッケージ.csv」として保存し，このノートブック上にアップロードしよう．

※「3-4. ディープニューラルネットワーク（DNN）」，「3-4. 学習用データと学習済みモデル」，「3-6. 決定木」，「3-6. サポートベクターマシン」，「3-7. 形態素解析」，「3-7. 表現学習」と同じデータセットを作成するため，もし同じものを持っている場合は以下の取得作業は不要なので，そちらをアップロードしよう．

```
PREFIX ma: <https://mediaarts-db.bunka.go.jp/data/property#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?発行者 ?プラットフォーム ?レーティング ?公開年月日
WHERE {
  ?アイテム
        schema:genre "ゲームパッケージ";
        schema:contentRating ?レーティング;
        schema:gamePlatform ?プラットフォーム;
        schema:publisher ?発行者;
        rdfs:label ?ラベル;
        ma:datePublished ?公開年月日.
}
```

取得したデータセットを，Python上で扱いやすい形にしていく． pandasというライブラリを用いて，csv形式のファイルの中身をDataFrameという型に変換し，扱っていく．

In [8]:
```
# 必要なモジュールのインポート
!pip install pykakasi
import pykakasi
import pandas as pd
import collections
```

```
Requirement already satisfied: pykakasi in /usr/local/lib/python3.10/dist-packages (2.2.1)
Requirement already satisfied: jaconv in /usr/local/lib/python3.10/dist-packages (from pykakas
i) (0.3.4)
Requirement already satisfied: deprecated in /usr/local/lib/python3.10/dist-packages (from pyk
akasi) (1.2.14)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (fro
m deprecated->pykakasi) (1.14.1)
```

In [9]:
```
games = pd.read_csv("/content/ゲームパッケージ.csv")
games = games.drop_duplicates(subset='ラベル')
games = games.reset_index(drop=True)
games
```

Out[9]:

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
0	Dance Dance Revolution GB2	コナミ株式会社	ゲームボーイ	CERO D (17才以上対象)	2000-11-16
1	プロ野球 ファミスタ リターンズ	株式会社バンダイナムコエンターテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2015-10-08
2	プロ野球 ファミスタ 2011	株式会社バンダイナムコゲームス	ニンテンドー3DS	CERO A (全年齢対象)	2011-03-31
3	フロッガー3D	株式会社コナミデジタルエンタテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2011-09-22
4	ブレイブリーセカンド	株式会社スクウェア・エニックス	ニンテンドー3DS	CERO C (15才以上対象)	2015-04-23
...	...	...	...	...	...
19452	バイオショック インフィニット PlayStation Now版	テイクツー・インタラクティブ・ジャパン合同会社	プレイステーション Now	CERO D (17才以上対象)	2017-09-19
19453	バイオハザード リベレーションズ アンベールド エディション PlayStation Now版	株式会社カプコン	プレイステーション Now	CERO D (17才以上対象)	2016-09-20
19454	巫剣神威控 PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO C (15才以上対象)	2017-07-20
19455	ネバーエンディング ナイトメア PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO D (17才以上対象)	2017-07-20
19456	雷電IV OverKill PlayStation Now版	モス	プレイステーション Now	CERO A (全年齢対象)	2017-02-21

19457 rows × 5 columns

かな漢字変換

今回はkakasiというモジュールを利用する.

In [10]:
```
# kakasiのインスタンスを作成
kks = pykakasi.kakasi()

# 変換
result = kks.convert(games['ラベル'][0])
print(result)
```

```
[{'orig': 'Dance Dance Revolution GB2', 'hira': 'Dance Dance Revolution GB2', 'kana': 'Dance Dance Revolution GB2', 'hepburn': 'Dance Dance Revolution GB2', 'kunrei': 'Dance Dance Revolution GB2', 'passport': 'Dance Dance Revolution GB2'}]
```

上のように，オリジナルの文字列からひらがな・カタカナ・ローマ字などへの変換結果が出ている． もしひらがなの変換結果のみを受け取りたい場合は，次のように `'hira'` を指定して取得する．

In [11]:
```
for token in result:
  print(token['hira'])
```

```
Dance Dance Revolution GB2
```

In [12]:
```
# 全データに対してかな漢字変換を行ってみる

# ひらがなのデータを入れる
title_hira = []
# オリジナルのデータを入れる
title_origin = []

for title in games['ラベル']:
  title_hira.extend([token["hira"] for token in kks.convert(title)])
  title_origin.extend([token["orig"] for token in kks.convert(title)])
```

この結果，出てくる単語の頻度がどのように変化するのか確認してみよう．

In [13]:
```
count_hira = collections.Counter(title_hira)
count_origin = collections.Counter(title_origin)

print(f"ひらがなに変換した後の「さくら」の登場回数: {count_hira['さくら']}")
print(f"オリジナルの「さくら」の登場回数: {count_origin['さくら']}")
print(f"オリジナルの「サクラ」の登場回数: {count_origin['サクラ']}")
print(f"オリジナルの「桜」の登場回数: {count_origin['桜']}")
```

```
ひらがなに変換した後の「さくら」の登場回数: 123
オリジナルの「さくら」の登場回数: 8
オリジナルの「サクラ」の登場回数: 36
オリジナルの「桜」の登場回数: 79
```

このように表記の異なるものについても同じように集計することができる． しかし，同音異義語についても同じように集計されてしまうため，注意が必要である．

In [14]:
```
print(f"ひらがなに変換した後の「じゅう」の登場回数: {count_hira['じゅう']}")
print(f"オリジナルの「銃」の登場回数: {count_origin['銃']}")
print(f"オリジナルの「十」の登場回数: {count_origin['十']}")
print(f"オリジナルの「重」の登場回数: {count_origin['重']}")
```

```
ひらがなに変換した後の「じゅう」の登場回数: 11
オリジナルの「銃」の登場回数: 1
オリジナルの「十」の登場回数: 6
オリジナルの「重」の登場回数: 4
```

3-7. 言語・知識 - 形態素解析、単語分割、係り受け解析

テキストを「トークン」と呼ばれる表現要素の最小単位の集合に分割することを分かち書き（単語分割）と呼び，テキスト（単位として文）をトークンに分割し，各トークンに品詞や語形・活用形などの情報を付与する処理を形態素解析と呼ぶ．また，それら単語の関係を係り受け構造（依存構造）と呼び，この構造を解析することを係り受け解析という．

実用上ではこれらの処理をどのようにして行っていくのか，ゲームパッケージのタイトルを利用して具体的にみてみよう．

データの取得・整形

メディア芸術データベース・ラボ上で提供されているSPARQLクエリサービスを利用して，ゲームパッケージのデータを取得していく．

以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる．

取得したCSVファイルを「ゲームパッケージ.csv」として保存し，このノートブック上にアップロードしよう．

※「3-4. ディープニューラルネットワーク（DNN）」，「3-4. 学習用データと学習済みモデル」，「3-6. 決定木」，「3-6. サポートベクターマシン」，「3-7. かな漢字変換」，「3-7. 表現学習」と同じデータセットを作成するため，もし同じものを持っている場合は以下の取得作業は不要なので，そちらをアップロードしよう．

```
PREFIX ma: <https://mediaarts-db.bunka.go.jp/data/property#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?発行者 ?プラットフォーム ?レーティング ?公開年月日
WHERE {
  ?アイテム
        schema:genre "ゲームパッケージ";
        schema:contentRating ?レーティング;
        schema:gamePlatform ?プラットフォーム;
        schema:publisher ?発行者;
        rdfs:label ?ラベル;
        ma:datePublished ?公開年月日.
}
```

取得したデータセットを，Python上で扱いやすい形にしていく． pandasというライブラリを用いて，csv形式のファイルの中身をDataFrameという型に変換し，扱っていく．

In [1]:
```
# 必要なモジュールのインポート
!pip install mecab-python3
!pip install unidic-lite
!pip install pyknp
import MeCab
from pyknp import KNP
from pyknp import Juman
```

```
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
import re
```

```
Collecting mecab-python3
  Downloading mecab_python3-1.0.8-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (581 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 581.7/581.7 kB 9.4 MB/s eta 0:00:00a 0:00:01
Installing collected packages: mecab-python3
Successfully installed mecab-python3-1.0.8
Collecting unidic-lite
  Downloading unidic-lite-1.0.8.tar.gz (47.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.4/47.4 MB 17.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: unidic-lite
  Building wheel for unidic-lite (setup.py) ... done
  Created wheel for unidic-lite: filename=unidic_lite-1.0.8-py3-none-any.whl size=47658817 sha256=78d4500ae627f9f5f21a2de8e777e3ea7c62362df89a614b187f3d3f3fd24e71
  Stored in directory: /root/.cache/pip/wheels/89/e8/68/f9ac36b8cc6c8b3c96888cd57434abed96595d444f42243853
Successfully built unidic-lite
Installing collected packages: unidic-lite
Successfully installed unidic-lite-1.0.8
Collecting pyknp
  Downloading pyknp-0.6.1-py3-none-any.whl (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.5/42.5 kB 1.5 MB/s eta 0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from pyknp) (1.16.0)
Installing collected packages: pyknp
Successfully installed pyknp-0.6.1
```

In [2]:
```
games = pd.read_csv("/content/ゲームパッケージ.csv")
games = games.drop_duplicates(subset='ラベル')
games = games.reset_index(drop=True)
games
```

Out[2]:

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
0	Dance Dance Revolution GB2	コナミ株式会社	ゲームボーイ	CERO D (17才以上対象)	2000-11-16
1	プロ野球 ファミスタ リターンズ	株式会社バンダイナムコエンターテイメント	ニンテンドー3DS	CERO A (全年齢対象)	2015-10-08
2	プロ野球 ファミスタ 2011	株式会社バンダイナムコゲームス	ニンテンドー3DS	CERO A (全年齢対象)	2011-03-31
3	フロッガー3D	株式会社コナミデジタルエンタテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2011-09-22
4	ブレイブリーセカンド	株式会社スクウェア・エニックス	ニンテンドー3DS	CERO C (15才以上対象)	2015-04-23

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
...	...	...	...	...	...
19452	バイオショック インフィニット PlayStation Now版	テイクツー・インタラクティブ・ジャパン合同会社	プレイステーション Now	CERO D (17才以上対象)	2017-09-19
19453	バイオハザード リベレーションズ アンベールド エディション PlayStation Now版	株式会社カプコン	プレイステーション Now	CERO D (17才以上対象)	2016-09-20
19454	巫剣神威控 PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO C (15才以上対象)	2017-07-20
19455	ネバーエンディング ナイトメア PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO D (17才以上対象)	2017-07-20
19456	雷電IV OverKill PlayStation Now版	モス	プレイステーション Now	CERO A (全年齢対象)	2017-02-21

19457 rows × 5 columns

単語分割（分かち書き）

英語の文章は「I have a pen.」のように単語はスペースで区切られているのに対し，日本語の文章は「私はペンを持っている．」のように単語が全て連結した文章になっているため，スペースで区切るような手法で単語分割をすることが難しい．

そこで，こういった日本語データに対する分かち書きをするためのモジュールがいくつか用意されているため，それを利用して分かち書きを行ってみよう．

今回はMeCabというモジュールを利用する．

In [3]:
```
# 分かち書きを行うインスタンスwakatiの作成
wakati = MeCab.Tagger('-Owakati')

# 試しに,ゲームタイトルを一つ分かち書きにしてみる
# strip()で最後の改行文字列を削除している
wakati.parse(games['ラベル'][0]).strip()
```

Out[3]: 'Dance Dance Revolution GB 2'

このように日本語の文章をスペース区切りの文字列に変換することができる． これを全てのゲームタイトルに適用するとこのようになる．

In [4]:
```
# 分かち書きしたタイトルを格納するリスト
wakati_titles = []

for title in games['ラベル']:
  wakati_titles.append(wakati.parse(title).strip())

# 最初から5個目まで見てみる
wakati_titles[:5]
```

Out[4]:
```
['Dance Dance Revolution GB 2',
 'プロ 野球 ファミスタ リターン ズ',
 'プロ 野球 ファミスタ 2011',
 'フロッガー 3 D',
 'ブレイブリーセカンド']
```

形態素解析

単語分割（分かち書き）と同じように，形態素解析もMeCabを利用して行うことができる．

In [5]:
```
# 試しに,ゲームタイトルを一つ形態素解析してみる
node = wakati.parseToNode(games['ラベル'][0])

while node:
  word = node.surface
  word_class = node.feature
  print("トークン: ", word)
  print("品詞: ", word_class)
  print()
  node = node.next
```

```
トークン:
品詞: BOS/EOS,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*

トークン: Dance
品詞: 名詞,普通名詞,一般,*,*,*

トークン: Dance
品詞: 名詞,普通名詞,一般,*,*,*

トークン: Revolution
品詞: 名詞,普通名詞,一般,*,*,*

トークン: GB
品詞: 名詞,普通名詞,一般,*,*,*

トークン: 2
品詞: 名詞,数詞,*,*,*,*

トークン:
品詞: BOS/EOS,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*
```

ここで，最初と最後に現れる「BOS/EOS」というのは自然言語などを扱う時にはよく出てくるもので，「Begin Of Sentence」，「End Of Sentence」のことであり，文章のはじめと終わりを表す特別なトークンである．これらは文章を生成したりする時には学習が必要だが，不要な時には以下のようにループ文を工夫することで取り除くこともできる．

In [6]:
```
# BOS/EOSを無視するバージョン
node = wakati.parseToNode(games['ラベル'][0])
node = node.next
while node.next:
  word = node.surface
  word_class = node.feature
  print("トークン: ", word)
  print("品詞: ", word_class)
  print()
  node = node.next
```

トークン： Dance
品詞： 名詞,普通名詞,一般,*,*,*

トークン： Dance
品詞： 名詞,普通名詞,一般,*,*,*

トークン： Revolution
品詞： 名詞,普通名詞,一般,*,*,*

トークン： GB
品詞： 名詞,普通名詞,一般,*,*,*

トークン： 2
品詞： 名詞,数詞,*,*,*,*

全てのゲームタイトルに適用する場合は次のようになる.

In [7]:
```
# 形態素解析したタイトルを格納するリスト
hinshi = []

for title in games['ラベル']:
  node = wakati.parseToNode(title)
  node = node.next
  # 形態素解析後のタイトルを[[トークン,品詞], [トークン,品詞], ...]のように保存するリスト
  title_hinshi = []
  while node.next:
    word = node.surface
    word_class = node.feature
    title_hinshi.append([word, word_class])
    node = node.next
  hinshi.append(title_hinshi)

# 最初から5個目まで見てみる
for title in hinshi[:5]:
  for token in title:
    print(token[0])
    print(token[1])
    print()
  print("==========================================")
```

Dance
名詞,普通名詞,一般,*,*,*

Dance
名詞,普通名詞,一般,*,*,*

Revolution
名詞,普通名詞,一般,*,*,*

GB
名詞,普通名詞,一般,*,*,*

2
名詞,数詞,*,*,*,*

==========
プロ
名詞,普通名詞,一般,*,*,*,プロ,プロ-pro,プロ,プロ,プロ,プロ,外,*,*,*,*,プロ,プロ,プロ,プロ,*,*,1,C1,*

野球

名詞,普通名詞,一般,*,*,*,ヤキュウ,野球,野球,ヤキュー,野球,ヤキュー,漢,*,*,*,*,ヤキュウ,ヤキュウ,ヤキュウ,ヤキュウ,*,*,0,C2,*

ファミスタ
名詞,普通名詞,一般,*,*,*

リターン
名詞,普通名詞,サ変可能,*,*,*,リターン,リターン-return,リターン,リターン,リターン,リターン,外,*,*,*,*,リターン,リターン,リターン,リターン,*,*,2,C1,*

ズ
記号,一般,*,*,*,*,ズ,ズ,ズ,ズ,ズ,ズ,記号,*,*,*,*,ズ,ズ,ズ,ズ,*,*,1,*,*

==========
プロ
名詞,普通名詞,一般,*,*,*,プロ,プロ-pro,プロ,プロ,プロ,プロ,外,*,*,*,*,プロ,プロ,プロ,プロ,*,*,1,C1,*

野球
名詞,普通名詞,一般,*,*,*,ヤキュウ,野球,野球,ヤキュー,野球,ヤキュー,漢,*,*,*,*,ヤキュウ,ヤキュウ,ヤキュウ,ヤキュウ,*,*,0,C2,*

ファミスタ
名詞,普通名詞,一般,*,*,*

2011
名詞,数詞,*,*,*,*

==========
フロッガー
名詞,固有名詞,一般,*,*,*,フロッガー,フロッガー,フロッガー,フロッガー,フロッガー,フロッガー,固,*,*,*,*,フロッガー,フロッガー,フロッガー,フロッガー,*,*,2,*,*

3
名詞,数詞,*,*,*,*

D
名詞,普通名詞,一般,*,*,*

==========
ブレイブリーセカンド
名詞,普通名詞,一般,*,*,*

==========

3-7. 言語・知識 - 表現学習（エンベディング）

ある情報を数値のベクトルで表現することを埋め込みやembedding（エンベディング）と呼ぶ． 特に，自然言語を機械学習で扱おうとするとき，文字列をどう表現するかというのは大きな問題の一つになる．

代表的には，One-hotベクトルによる表現や，Word2Vecを用いた表現などが存在する．実用上ではこれらの処理をどのようにして行っていくのか，ゲームパッケージのタイトルを利用して具体的にみてみよう．

データの取得・整形

メディア芸術データベース・ラボ上で提供されているSPARQLクエリサービスを利用して，ゲームパッケージのデータを取得していく．

以下のコードをSPARQLクエリサービス上に入力して実行する，または入力を省略したこちらのURLから実行してデータを取得できる． 取得したCSVファイルを「ゲームパッケージ.csv」として保存し，このノートブック上にアップロードしよう．

※「3-4. ディープニューラルネットワーク（DNN）」，「3-4. 学習用データと学習済みモデル」，「3-6. 決定木」，「3-6. サポートベクターマシン」，「3-7. 形態素解析」，「3-7. かな漢字変換」と同じデータセットを作成するため，もし同じものを持っている場合は以下の取得作業は不要なので，そちらをアップロードしよう．

```
PREFIX ma: <https://mediaarts-db.bunka.go.jp/data/property#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/> SELECT ?ラベル ?発行者 ?プラットフォーム ?レーティング ?公開年月日
WHERE {
  ?アイテム
        schema:genre "ゲームパッケージ";
        schema:contentRating ?レーティング;
        schema:gamePlatform ?プラットフォーム;
        schema:publisher ?発行者;
        rdfs:label ?ラベル;
        ma:datePublished ?公開年月日.
}
```

取得したデータセットを，Python上で扱いやすい形にしていく． pandasというライブラリを用いて，csv形式のファイルの中身をDataFrameという型に変換し，扱っていく．

In [1]:
```
# 必要なモジュールのインポート
!pip install pykakasi
!pip install --upgrade gensim

from gensim.models import Word2Vec
import pykakasi
import pandas as pd
import numpy as np
```

```
import collections
from sklearn.preprocessing import OneHotEncoder
```

```
Collecting pykakasi
  Downloading pykakasi-2.2.1-py3-none-any.whl (2.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.4/2.4 M
B 11.0 MB/s eta 0:00:00
Collecting jaconv (from pykakasi)
  Downloading jaconv-0.3.4.tar.gz (16 kB)
  Preparing metadata (setup.py) ... done
Collecting deprecated (from pykakasi)
  Downloading Deprecated-1.2.14-py2.py3-none-any.whl (9.6 kB)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (fro
m deprecated->pykakasi) (1.14.1)
Building wheels for collected packages: jaconv
  Building wheel for jaconv (setup.py) ... done
  Created wheel for jaconv: filename=jaconv-0.3.4-py3-none-any.whl size=16415 sha256=165a2
eff172cfb166e1e787bcde455a4f04f190d6eedeb01397eb34d3f07bb70
  Stored in directory: /root/.cache/pip/wheels/46/8f/2e/a730bf1fca05b33e532d5d91dabdf40
6c9b718ec85b01b1b54
Successfully built jaconv
Installing collected packages: jaconv, deprecated, pykakasi
Successfully installed deprecated-1.2.14 jaconv-0.3.4 pykakasi-2.2.1
Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages (4.3.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from
gensim) (1.25.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from ge
nsim) (1.11.4)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.10/dist-packages (f
rom gensim) (6.4.0)
```

In [2]:
```
games = pd.read_csv("/content/ゲームパッケージ.csv")
games = games.drop_duplicates(subset='ラベル')
games = games.reset_index(drop=True)
games
```

Out[2]:

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
0	Dance Dance Revolution GB2	コナミ株式会社	ゲームボーイ	CERO D (17才以上対象)	2000-11-16
1	プロ野球 ファミスタ リターンズ	株式会社バンダイナムコエンターテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2015-10-08
2	プロ野球 ファミスタ 2011	株式会社バンダイナムコゲームス	ニンテンドー3DS	CERO A (全年齢対象)	2011-03-31
3	フロッガー3D	株式会社コナミデジタルエンタテインメント	ニンテンドー3DS	CERO A (全年齢対象)	2011-09-22
4	ブレイブリーセカンド	株式会社スクウェア・エニックス	ニンテンドー3DS	CERO C (15才以上対象)	2015-04-23
...	...	...	...	...	...

	ラベル	発行者	プラットフォーム	レーティング	公開年月日
19452	バイオショック インフィニット PlayStation Now版	テイクツー・インタラクティブ・ジャパン合同会社	プレイステーション Now	CERO D (17才以上対象)	2017-09-19
19453	バイオハザード リベレーションズ アンベールド エディション PlayStation Now版	株式会社カプコン	プレイステーション Now	CERO D (17才以上対象)	2016-09-20
19454	巫剣神威控 PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO C (15才以上対象)	2017-07-20
19455	ネバーエンディング ナイトメア PlayStation Now版	アクティブゲーミングメディア	プレイステーション Now	CERO D (17才以上対象)	2017-07-20
19456	雷電IV OverKill PlayStation Now版	モス	プレイステーション Now	CERO A (全年齢対象)	2017-02-21

19457 rows × 5 columns

One-Hotベクトル

未知の単語が出てきた順にインデックスを振り，対応するインデックスの要素だけ1，あとは0というようなベクトルを作成する． pykakasiというモジュールを用いて日本語の文章を単語に分割し，ワンホットベクトルにしていく．

In [3]:
```
# pykakasiのインスタンスを用意
kks = pykakasi.kakasi()

# ゲームタイトルに出てくる単語を順番に全て格納するリスト
words = []

for title in games['ラベル']:
  words.extend([token["orig"] for token in kks.convert(title)])

# ワンホットベクトルへのエンコードを行うインスタンスの作成
onehot_encoder = OneHotEncoder(sparse = False,dtype = int)

# ワンホットベクトルへの学習・変換を同時に行う
words_embedded = onehot_encoder.fit_transform(np.array(words).reshape(-1,1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

「パッケージ」という単語のベクトルを見てみよう．

In [4]:
```
# 出力が省略されないように閾値を変えておく
np.set_printoptions(threshold=100000)

input = "パッケージ"
print("onehotベクトルの次元数: ", len(words_embedded[0]))
print("元の単語: ", input)
```

```python
print("ワンホットベクトル: ", onehot_encoder.transform([[input]]))

# 閾値を元に戻す
np.set_printoptions(threshold=1000)
```

```
onehotベクトルの次元数: 20323
元の単語: パッケージ
ワンホットベクトル: [[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

One-Hotベクトル表現は，語彙数とベクトルの次元数が同じになってしまうという点で，計算効率の悪さや入力としての扱いづらさ（入力次元数が定まらないと機械学習がしづらい）などの問題がある．

Word2Vec

One-Hotベクトルの欠点を克服するのがこのWord2Vecである．これは単語同士の意味の近さをベクトルの大きさや方向の近さで表現する手法であり，指定した次元に埋め込むこともできるため，効率的に学習が行える． Word2Vecモデルの学習を実際にやってみよう．

In [5]:
```
# ゲームタイトルを単語を分割したリストを順番に全て格納するリスト
words = []

for title in games['ラベル']:
  words.append([token["orig"] for token in kks.convert(title)])
```

In [6]:
```
# Word2Vecモデルを学習
model = Word2Vec(words,  sg=1, vector_size=100, window=3, min_count=0) # 引数sg=1でskip
```

In [7]:
```
# "パッケージ"のベクトル表現を出力
input = "パッケージ"
result = model.wv.get_vector(input)
print("onehotベクトルの次元数: ", len(result))
print("元の単語: ", input)
print("ベクトル: ", result)
```

```
onehotベクトルの次元数:  100
元の単語:  パッケージ
ベクトル:  [-0.11668929  0.09766782  0.3968426   0.6236199  -0.34263754 -0.26765057
  0.36841166  0.39545757 -0.7935258  -0.13745552 -0.153762   -0.2411714
 -0.45867485  0.35353434 -0.05605572 -0.17603661  0.2606729   0.25922826
 -0.4332906  -0.8751171   0.38124487 -0.2746471   0.9497776  -0.31293562
 -0.26312435 -0.18485972 -0.50300866  0.42192176 -0.6822334   0.31044954
  0.24615403 -0.22320043 -0.18758088 -0.69846314 -0.28777418  0.10628433
  0.00291802  0.05891388  0.08581531 -0.21064182  0.34355447 -0.56263196
 -0.5777083   0.06062189  0.33344656  0.0877355  -0.38855493  0.18607187
  0.5731279   0.18241204 -0.02902607 -0.06122071  0.21697947 -0.24158446
 -0.3627133  -0.11245427 -0.04041747 -0.4449694  -0.37351665  0.17147596
  0.3219174  -0.23757496  0.18068212  0.031368    0.207157    0.35256016
 -0.06172541  0.6798512  -0.6101884  -0.11404646 -0.04860069  0.46427938
  0.2708793  -0.11087355  0.4998917   0.11074626  0.27610767  0.33965242
  0.09810583 -0.59083843 -0.7763726  -0.5032251  -0.07470347  0.32645842
 -0.5807437  -0.7155814   0.49727964  0.19476798 -0.08383452 -0.01151026
  0.42493385  0.13046499 -0.28286368 -0.2634584   0.4620586   0.22640374
  0.18026938 -0.33898634 -0.07521158  0.03934561]
```

Word2Vecの特徴として，ベクトルの近さと単語の近さに関係があるという点がある． 実際に「パッケージ」に近いベクトルを出力してみよう．

In [8]:
```
# "パッケージ"に近しい意味を持つ単語を出力
for i in model.wv.most_similar('パッケージ'):
    print(i)
```

```
('ベスト', 0.9795853495597839)
('同梱', 0.9794086217880249)
(' PlayStation Now', 0.9755346775054932)
('替え', 0.9713441729545593)
('\u3000', 0.9557426571846008)
('吹き', 0.9538519978523254)
('ダウンロード', 0.9515132308006287)
(' of LOVE!', 0.951181948184967)
('プラチナコレクション', 0.9496081471443176)
('デラックス', 0.9489535689353943)
```

「パッケージ」に近い意味の単語を出力させてみると，「パッケージ版」のようにゲームタイトルの最後に「~~版」という形で載っていそうな単語が多く出力されていることがわかる．

この手法の欠点としては，きちんと単語同士の距離をベクトルに反映させるためには十分な量の学習データが必要であるという点である．今回の「ゲームタイトル」のようにあまり文章的ではない学習データでは，直感的にはわかりづらい結果が出てくることもある． そのため，利用するデータの大きさや種類によって使い分けられると良い．