Pandas Document

1. What is tidy data

What is tidy data : Definition of tidy data

2. Options

```
# max column views
pd.options.display.max_columns = 99

# suppress scientific notation
pd.set_option('display.float_format', lambda x: '%.5f' % x)
```

3. Data Reshaping

3.1 group by

When we want to make data arranged by groupby() key, we use groupby().

```
df.groupby('key1').sum()
df.groupby(['key1', 'key2']).sum()
```

After making groupby() operator() datasets, we need to unstack to make tidier dataset.

```
df.groupby(['key1']).sum().reset_index(inplace = True)
```

When we use as_indx = False parameter, we don't have to use ['col'] to extarct aggregated values.

3.2 melt()

By using melt, we make tidier datset storing columns as one column variable previous columns.

3.3 crosstab()

We usually use crosstab() for check hist chart of two columns.

3.4 stack() and unstack()

melt의 경우에는 column을 tidy한 데이터로 만들어 주기위해 column을 value로 끌어내려주는 역할을 수행하지만, stack()과 unstack()의 경우 다중 행, 열인덱스를 각각의 위치를 치환하는 행위를 수행한다. stack의 경우에는 열 인덱스를 행 인덱스로, unstack의 경우에는 그 역을 수행한다. stack().reset_index()의 경우 melt()와 동일한 매서드를 수행하게 된다.

4. Data Subsetting

4.1 Finding max and min value

if we want to get index or column having max or min value, we add 'idx' in front of max() and min() function.

```
ign_data.loc['PlayStation Vita', :].idxmin()
```

4.2 Series 와 Dataframe 추출 방식

- df['A'] : Series
- df[['A']] : Dataframe

4.3 .loc

We should use .loc[,] more than [dataframe][boolean indexing][columns] to make code more tidier.

```
# Good case
df.loc[boolean condition, 'column'] == 'ex1'

# Complicated case
df[boolean condition]['column'] == 'ex1'
```

4.4 .isin()

isin() method is usually used in making datasets fits boolean indexing for specific condition.
Find with long index, and variable fitting long condition.

```
df_academy_selected = df_academy[df_academy['상권업종소분류명'].isin(academy_count_1000.index)]

top_platforms = (
    df["Platform"].value_counts().sort_values(ascending=False).head(5).index.values # make perfect data frame
  or list
)


sns.boxplot(
    y="Platform",
    x="Critic_Score",
    data=df[df["Platform"].isin(top_platforms)],
    orient="h",
);
```

4.5 map for isin()

```
n_trop = reviews.description.map(lambda desc: "tropical" in desc).sum()
n_fruity = reviews.description.map(lambda desc: "fruity" in desc).sum()
descriptor_counts = pd.Series([n_trop, n_fruity], index=['tropical', 'fruity'])
```

4.6 .str.contains()

str().contains() method is usually used in finding datasets fit boolean indexing which fits the contians code.
Find with noncompleted words, and fitting small condition.

```
df_31.loc[df_31['상호명_소문자'].str.contains('배스킨|배스킨라빈스'), "브랜드명"] = "배스킨라빈스"
df_31.loc[~df_31['상호명_소문자'].str.contains('배스킨|배스킨|baskin'), "브랜드명"] = "배스킨라빈스"
```

5. Data mutating

5.1 rename()

rename can change column name what we want.

```
df_pre = df_pre.rename(columns = {'구분' : 'Category'})
```

5.2 replace()

replace() can change value what we want.

```
df_pre['Category'] = df_pre['Category'].replace({'전과자' : 'Old_hand', '초범' : 'First_hand'})
```

5.3 .str.split().str

If we want to make new column using previous columns having string values, we divide using str.split() and index by str.

```
data['First_name'] = data['Name'].str.split(",").str[0]
data.loc[data['Sex'] == 'male', 'First_name'].value_counts()

# Or use for loop to change column, but this way takes time more.
for i in range(df.shape[0]):
        df.loc[i, 'grp'] = str(df.loc[i, 'id']).split("_")[0]
```

5.4 .str.split().str.rstrip()

strip() option can delete left or right strip in condition.

```
df_time['Time'].str.split("(").str[1].str.rstrip(')')
```

5.5 order by categories

실수 값을 카테고리 값으로 변환

실수 값을 크기 기준으로 하이 카테고리 값으로 변환하고 싶을 때는 다음과 같은 명령을 사용한다.

- cut : 실수 값의 경계선을 지정하는 경우
- qcut : 경수가 똑같은 구간으로 나누는 경우 cut 명령어 반환하는 값은 Categorical 클래스 객체이다. 이 객체는 categores 속성으로 라벨 문자열을, codes 속성으로 정수로 인코딩한 카테고리 값을 가진다. 따라서 위 데이터프레임의 결과는 문자열이 아니고, 이를 문자열로 만드려면 astype 메서드를 사용해야 한다.

```
bins = [1, 20, 30, 50, 70, 100]
labels = ["이상년자", "청년", "중년", "장년", "노년"]
titanic['age_cat'] = pd.cut(titanic.age, bins = bins, labels = labels)
titanic.head()

titanic['category3'] = pd.cut(titanic.age, bins = bins, labels = labels)
titanic['category3'] = titanic.apply(lambda x : "이상년자" if x.age < 20 else titanic.category3.astype('str') +
x.sex, axis = 1)
titanic.head()
```

고정된 카테고리로 정렬

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

        data = {
            'id': [2967, 5335, 13950, 6141, 6169],
            'Player': ['Cedric Hunter', 'Maurice Baker',
                       'Ratko Varda' ,'Ryan Bowen' ,'Adrian Caldwell'],
            'Year': [1991, 2004, 2001, 2009, 1997],
            'Age': [27, 25, 22, 34, 31],
            'Tm': ['CHH', 'VAN', 'TOT', 'OKC', 'DAL'],
            'G': [6, 7, 60, 52, 81]
        }

        # Create DataFrame
        df = pd.DataFrame(data)

        # Define the sorter
        sorter = ['TOT', 'ATL', 'BOS', 'BRK', 'CHA', 'CHH', 'CHI', 'CLE', 'DAL', 'DEN',
                  'DET', 'GSW', 'HOU', 'IND', 'LAC', 'LAL', 'MEM', 'MIA', 'MIL',
                  'MIN', 'NJN', 'NOH', 'NOK', 'NOP', 'NYK', 'OKC', 'ORL', 'PHI',
                  'PHO', 'POR', 'SAC', 'SAS', 'SEA', 'TOR', 'UTA', 'VAN', 'WAS', 'WSB']

        # With the data-frame and sorter, which is a category-order, we can do the following in pandas 15.1:

        # Convert Tm-column to category and in set the sorter as categories hierarchy
        # Youc could also do both lines in one just appending the cat.set_categories()
        df.Tm = df.Tm.astype("category")
        df.Tm.cat.set_categories(sorter, inplace=True)

        print(df.Tm)

        df.sort_values(["Tm"])  ## 'sort' changed to 'sort_values'

0   CHH
1   VAN
2   TOT
3   OKC
4   DAL
Name: Tm, dtype: category
Categories (38, object): ['TOT', 'ATL', 'BOS', 'BRK', ..., 'UTA', 'VAN', 'WAS', 'WSB']

Out[1]:
```

	id	Player	Year	Age	Tm	G
2	13950	Ratko Varda	2001	22	TOT	60
0	2967	Cedric Hunter	1991	27	CHH	6
4	6169	Adrian Caldwell	1997	31	DAL	81
3	6141	Ryan Bowen	2009	34	OKC	52
1	5335	Maurice Baker	2004	25	VAN	7

5.6 Grouping by bins

```
# categorize age features
bins = [20, 30, 40, 50, 60, 70, 80]
labels = ['20~30', '30~40', '40~50', '50~60', '60~70', '70~80']
df_raw['age_cat'] = pd.cut(df_raw.Age, bins = bins, labels = labels)
```

5.7 apply function method

map()

map() must used in Series type.

data = {'team' : ['russia', 'saudiarabia', 'egypt', 'uruguay'], 'against': ['saudiarabia', 'russia', 'uruguay', 'egypt'], 'fifa_rank': [65, 63, 31, 21]} columns = ['team', 'against', 'fifa_rank'] df = pd.DataFrame(data, columns = columns) def total_record(team): ... # calculation from Database ... return win_count, draw_count, lose_count, winning_rate df['winning_rate'] = df['team'].map(lambda x : total_record(x)[3])

apply()

apply() is used in Dataframe and Series type when we apply function with multiple columns. We use apply() to apply function directly to dataframe.

```
df.apply(function, axis = 1) # --> apply function directly
df.apply(lambda x : ) #--> make function and apply it directly

df['winning_rate'] = df.apply(lambda x:relative_record(x['team'], x['against'])[3], axis=1)
```

apply function to make multiple column and combine them

```
def add_Total(df) :
    return df[[col for col in df.columns if "Num" in col]].apply(np.sum, axis = 1)

def add_Num_per_Acc(df) :
    return round(add_Total(df) / df['Traffic_Accident'], 2)

def Num_critical(df) :
    return round(df['Num_of_Deaths'] / df['Traffic_Accident'], 3)

def apply_and_concat(df) :
    res = pd.DataFrame({'Total' : add_Total(df), 'Num_per_Acc' : add_Num_per_Acc(df), 'Critical' : Num_critica
l(df)})
    return pd.concat([df, res], axis = 1)

df_accident_month = apply_and_concat(df_accident_month)
df_attacker = apply_and_concat(df_attacker)

df_attacker.head()
```

6. Multivariate analysis

6.1 groupby()

3개 이상의 feature에 대하여 타겟 변수간의 상관정도를 나타내주기 위해 groupby와 visualization을 적절히 이용한다.

```
def rate(x) :
    return round(x.mean()*100, 2)
titanic.groupby(['sex', 'pclass', 'age_cat'])['survived'].agg(rate)
```

7. SQLite

7.1 Load file on SQLite

```
import sqlite3
conn = sqlite3.connect('crunchbase.db')

invest_iter = pd.read_csv('crunchbase-investments.csv', chunksize = 5000, encoding = 'ISO-8859-1',
                          parse_dates = ['funded_at'], usecols = col_use)

for chunk in invest_iter :
    chunk.to_sql('investment_table', conn, if_exists = 'append', index = False)
```

7.2 Retrieve data from SQLite

```
# Which category of company attracted the most investments?
query = """
    SELECT company_category_code, AVG(raised_amount_usd) as 'Average dollars'
        FROM investment_table
    GROUP BY company_category_code
    ORDER BY 2 DESC
    LIMIT 10;
"""
result_iter = pd.read_sql(query, conn, chunksize = 5000)
for chunk in result_iter :
    print(chunk)
```