Working with data

Anh Hao Nguyen Si

Hanoi Medical University

Working with data

'Experts often possess more data than judgment.'

—Colin Powell

Outline

1. Data cleaning and preparation with Pandas 
2. Basic Data Manipulation with SK-Learn 

3. Practice with a dataset

Data cleaning and preparation with Pandas 

GIGO

Outline

Introduction

The pandas (PANel + DAta) Python library is an open source allows for easy and fast data analysis and manipulation tools.

Pandas library providing numerical tables and time series data structures called DataFrame and Series, respectively.

Pandas was created to do the following:

- Provide data structures that can handle both time and non-time series data
- Allow mathematical operations on the data structures, ignoring the metadata of the data structures
- Use relational operations like those found in programming languages like SQL (join, group by, etc.)
- Handle missing data

Outline

Creating and describing data frame

```
>>> import pandas as pd
>>> df=pd.read_csv("https://www.openml.org/data/get_csv/37/dataset_37_diabetes.csv")
>>> # show first two rows
>>> print (df.head(2))
   preg  plas  pres  skin  insu  mass   pedi  age            class
0     6   148    72    35     0  33.6  0.627   50  tested_positive
1     1    85    66    29     0  26.6  0.351   31  tested_negative
>>> # show statistics
>>> df.describe()
             preg        plas        pres  ...        mass        pedi         age
count  768.000000  768.000000  768.000000  ...  768.000000  768.000000  768.000000
mean     3.845052  120.894531   69.105469  ...   31.992578    0.471876   33.240885
std      3.369578   31.972618   19.355807  ...    7.884160    0.331329   11.760232
min      0.000000    0.000000    0.000000  ...    0.000000    0.078000   21.000000
25%      1.000000   99.000000   62.000000  ...   27.300000    0.243750   24.000000
50%      3.000000  117.000000   72.000000  ...   32.000000    0.372500   29.000000
75%      6.000000  140.250000   80.000000  ...   36.600000    0.626250   41.000000
max     17.000000  199.000000  122.000000  ...   67.100000    2.420000   81.000000

[8 rows x 8 columns]
```

Selecting Rows Based on Conditionals

```
>>> # select top two rows where column 'preg'
>>> df[df['preg'] >=3].head(2)
   preg  plas  pres  skin  insu  mass   pedi  age            class
0     6   148    72    35     0  33.6  0.627   50  tested_positive
2     8   183    64     0     0  23.3  0.672   32  tested_positive
>>> # multiple conditions
>>> df[(df['preg'] >=3) & (df['age'] >= 65)]
     preg  plas  pres  skin  insu  mass   pedi  age            class
123     5   132    80     0     0  26.8  0.186   69  tested_negative
148     5   147    78     0     0  33.7  0.218   65  tested_negative
362     5   103   108    37     0  39.2  0.305   65  tested_negative
363     4   146    78     0     0  38.5  0.520   67  tested_positive
459     9   134    74    33    60  25.9  0.460   81  tested_negative
489     8   194    80     0     0  26.1  0.551   67  tested_negative
495     6   166    74     0     0  26.6  0.304   66  tested_negative
552     6   114    88     0     0  27.8  0.247   66  tested_negative
666     4   145    82    18     0  32.5  0.235   70  tested_positive
674     8    91    82     0     0  35.6  0.587   68  tested_negative
684     5   136    82     0     0   0.0  0.640   69  tested_negative
759     6   190    92     0     0  35.5  0.278   66  tested_positive
```

Renaming Columns and Finding the Mean, Variance and ...

- Renaming Columns

```
>>> df.rename(columns={'plas': 'glu', 'mass': 'bmi'}).head(2)
   preg  glu  pres  skin  insu   bmi   pedi  age            class
0     6  148    72    35     0  33.6  0.627   50  tested_positive
1     1   85    66    29     0  26.6  0.351   31  tested_negative
```

- Finding the Mean, Variance and ...

```
>>> print('Mean: {}'.format(df['age'].mean()))
Mean: 33.240885416666664
>>> print("Variance: {}".format(df['age'].var()))
Variance: 138.30304589037365
>>> print("Standard Deviation: {}".format(df['age'].std()))
Standard Deviation: 11.76023154067868
>>> print("Kurtosis: {}".format(df['age'].kurt()))
Kurtosis: 0.6431588885398942
>>> print("Skewness: {}".format(df['age'].skew()))
Skewness: 1.1295967011444805
```

Outline

Finding missing data

```
>>> import pandas as pd
>>> url="https://raw.githubusercontent.com/kinokoberuji/R-Tutorials/master/missingvalues.csv"
>>> df=pd.read_csv(url,sep = ";")
>>> df.head(2)
   Idx Outcome Var1 Var2  Var3  Var4 Var5
0    1    Dead    1  NaN   100  1175  NaN
1    2    Dead    0   90  #IND  1225   15
```


Pandas can recognize missing value codes like NA, NaN, empty,...


However it doesn't work with codes like -1,IND,999,***,...

```
>>> #Select 'Idx' in the data to make the index
>>> df=pd.read_csv(url,sep=";",index_col='Idx',na_values=['###','0','***','99999','IND','#IND'])
>>> # select missing values, show 2 rows
>>> df.head(2)
    Outcome  Var1  Var2   Var3  Var4 Var5
Idx
1      Dead   1.0   NaN  100.0  1175  NaN
2      Dead   NaN  90.0    NaN  1225   15
>>> df[df['Var1'].isnull()].head(2)
      Outcome  Var1  Var2  Var3  Var4 Var5
Idx
2        Dead   NaN  90.0   NaN  1225   15
3    Survived   NaN  90.0  90.0    na   15
```

Deleting a Data

```
>>> # Deleting a Column
>>> # axis=1 means the column axis
>>> df.drop('Var5', axis=1).head(2)
    Outcome  Var1  Var2   Var3  Var4
Idx
1      Dead   1.0   NaN  100.0  1175
2      Dead   NaN  90.0    NaN  1225
>>> # Deleting a Row
>>> # create new dataframe excluding the rows you want to delete
>>> df[df['Outcome'] != 'Dead'].head(2)
      Outcome  Var1  Var2  Var3 Var4  Var5
Idx
3    Survived   NaN  90.0  90.0   na    15
6    Survived   NaN   NaN  80.0  513  Null
>>> # Deleteing Observations with Missing Values
>>> df.dropna()
    Outcome  Var1  Var2  Var3 Var4 Var5
Idx
9      Dead   1.0  70.0  80.0  825   16
```

Read more:

Identifying the Three Types of Missing Data

Missing-Data Imputation

Filling In Missing Data

```
>>> df.fillna(0).head(2)
    Outcome  Var1  Var2   Var3  Var4 Var5
Idx
1      Dead   1.0   0.0  100.0  1175    0
2      Dead   0.0  90.0    0.0  1225   15
>>> df.fillna({'Var1': 0.5, 'Var2': 50}).head(2)
    Outcome  Var1  Var2   Var3  Var4 Var5
Idx
1      Dead   1.0  50.0  100.0  1175  NaN
2      Dead   0.5  90.0    NaN  1225   15
>>> df.fillna(method='ffill').head(2)
    Outcome  Var1  Var2   Var3  Var4 Var5
Idx
1      Dead   1.0   NaN  100.0  1175  NaN
2      Dead   1.0  90.0  100.0  1225   15
>>> df.fillna(df.mean()).head(2)
    Outcome  Var1       Var2        Var3  Var4 Var5
Idx
1      Dead   1.0  81.666667  100.000000  1175  NaN
2      Dead   1.4  90.000000   86.666667  1225   15
>>> _ = df.fillna(0, inplace=True)
>>> df.head(2)
    Outcome  Var1  Var2   Var3  Var4 Var5
Idx
1      Dead   1.0   0.0  100.0  1175    0
2      Dead   0.0  90.0    0.0  1225   15
```

Outline

Removing Duplicates

```
>>> data = pd.DataFrame(
... {'k1': ['one', 'two'] * 2+['two'],
...  'k2': [1, 1, 2, 3, 3]})
>>> data
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  two   3
>>> data.duplicated()
0    False
1    False
2    False
3    False
4     True
dtype: bool
>>> data.drop_duplicates()
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
```

```
>>> data.drop_duplicates(['k1'])
    k1  k2
0  one   1
1  two   1
>>> data.drop_duplicates(['k1', 'k2'],
...                      keep='last')
    k1  k2
0  one   1
1  two   1
2  one   2
4  two   3
```

Transforming data Use a Function or Mapping

```
>>> df=pd.read_csv("https://www.openml.org/data/get_csv/37/dataset_37_diabetes.csv")
>>> df.head(2)
   preg  plas  pres  skin  insu  mass   pedi  age            class
0     6   148    72    35     0  33.6  0.627   50  tested_positive
1     1    85    66    29     0  26.6  0.351   31  tested_negative
>>> def uppercase(x): return x.upper()
...
>>> df['class'].apply(uppercase)[0:2]
0    TESTED_POSITIVE
1    TESTED_NEGATIVE
Name: class, dtype: object
>>> # Applying a Function to Groups
>>> df.groupby('class').apply(lambda x: x.mean())
                    preg        plas  ...      pedi        age
class                                 ...
tested_negative  3.298000  109.980000  ...  0.429734  31.190000
tested_positive  4.865672  141.257463  ...  0.550500  37.067164

[2 rows x 8 columns]
>>> class_bi={'tested_negative': 0,'tested_positive': 1}
>>> df['binary'] = df['class'].map(class_bi)
>>> df.head(2)
   preg  plas  pres  skin  insu  mass   pedi  age            class  binary
0     6   148    72    35     0  33.6  0.627   50  tested_positive       1
1     1    85    66    29     0  26.6  0.351   31  tested_negative       0
```

Data transformation

- Replacing values

```
>>> df.replace({"preg":{1:"One",6:"Six"}}, inplace=True)
>>> df.head(2)
  preg  plas  pres  skin  insu  mass   pedi  age            class  binary
0  Six   148    72    35     0  33.6  0.627   50  tested_positive       1
1  One    85    66    29     0  26.6  0.351   31  tested_negative       0
```

- Discretization and Binning

```
>>> bins = [18, 25, 35, 60, 100]
>>> group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
>>> cat=pd.cut(df['age'], bins, labels=group_names)
>>> df = df.assign(cat=cat.values)
>>> df.head(2)
  preg  plas  pres  skin  insu  ...   pedi  age            class binary         cat
0  Six   148    72    35     0  ...  0.627   50  tested_positive      1  MiddleAged
1  One    85    66    29     0  ...  0.351   31  tested_negative      0  YoungAdult

[2 rows x 11 columns]
>>> cats = pd.qcut(df['age'], 4)
>>> pd.value_counts(cats)
(20.999, 24.0]    219
(29.0, 41.0]      200
(24.0, 29.0]      177
(41.0, 81.0]      172
Name: age, dtype: int64
```

Data transformation

- Detecting and Filtering Outliers

```
>>> import pandas as pd
>>> df=pd.read_csv("https://www.openml.org/data/get_csv/37/dataset_37_diabetes.csv")
>>> df[(df['preg']) > 10].head(2)
    preg  plas  pres  skin  insu  mass   pedi  age            class
24    11   143    94    33   146  36.6  0.254   51  tested_positive
28    13   145    82    19   110  22.2  0.245   57  tested_negative
>>> df['preg'][(df['preg'] > 10)]=11
>>> df[(df['preg']) > 10].head(2)
    preg  plas  pres  skin  insu  mass   pedi  age            class
24    11   143    94    33   146  36.6  0.254   51  tested_positive
28    11   145    82    19   110  22.2  0.245   57  tested_negative
```

- Computing Indicator/ Dummy Variables

```
>>> # A dummy variable is used in regression analysis to quantify categorical
>>> pd.get_dummies(pd.cut(df['age'], bins)).head(4)
   (18, 25]  (25, 35]  (35, 60]  (60, 100]
0         0         0         1          0
1         0         1         0          0
2         0         1         0          0
3         1         0         0          0
```

Outline

Hierarchical Indexing

```
>>> import numpy as np
>>> frame = pd.DataFrame(np.arange(12).reshape((4, 3)),
...                      index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
...                      columns=[['Ohio', 'Ohio', 'Colorado'],
...                      ['Green', 'Red', 'Green']])
>>> frame.index.names = ['key1', 'key2']
>>> frame.columns.names = ['state', 'color']
>>> frame
state      Ohio     Colorado
color     Green Red    Green
key1 key2
a    1        0   1        2
     2        3   4        5
b    1        6   7        8
     2        9  10       11
>>> frame.swaplevel('key1', 'key2').head(2) #Reordering and Sorting Levels
state      Ohio     Colorado
color     Green Red    Green
key2 key1
1    a        0   1        2
2    a        3   4        5
>>> frame.sum(level='key2') #Summary Statistics by Level
state  Ohio     Colorado
color Green Red    Green
key2
1         6   8       10
2        12  14       16
```

Read more: Hierarchical Indexing

Concate function

```
>>> df1 = pd.read_csv('C:/Users/AnhHao/Downloads/pandas/datacon1.csv')
>>> df2 = pd.read_csv('C:/Users/AnhHao/Downloads/pandas/datacon2.csv')
>>> df1.head(2)
           HoTen  Tuoi Gioitinh Chandoan   FEV1
0      Le Hong A    46      Nam     COPD  62.58
1  Nguyen Ngoc B    17       Nu      Hen  74.25
>>> df2.head(2)
      HoTen  Tuoi Gioitinh    Chandoan   FEV1
0  Ly Thi H    26       Nu  Binhthuong  90.47
1  Le Van T    23      Nam         Hen  83.51
>>> row_concat = pd.concat([df1, df2])
>>> row_concat.head(5)
           HoTen  Tuoi Gioitinh    Chandoan   FEV1
0      Le Hong A    46      Nam        COPD  62.58
1  Nguyen Ngoc B    17       Nu         Hen  74.25
2     Tran Van C    52      Nam        COPD  69.35
3   Nguyen Thi D    35       Nu  Binhthuong  83.96
0       Ly Thi H    26       Nu  Binhthuong  90.47
>>> new_case = pd.DataFrame([['Le Thi X', 'Nu', '30', '87.5']],
...                 columns = ['HoTen', 'Gioitinh', 'Tuoi', 'FEV1'])
>>> pd.concat([df1, new_case])
     Chandoan   FEV1 Gioitinh          HoTen Tuoi
0        COPD  62.58      Nam      Le Hong A   46
1         Hen  74.25       Nu  Nguyen Ngoc B   17
2        COPD  69.35      Nam     Tran Van C   52
3  Binhthuong  83.96       Nu   Nguyen Thi D   35
0         NaN   87.5       Nu       Le Thi X   30
```

Merge function

```
>>> dat1=pd.read_csv('C:/Users/AnhHao/Downloads/pandas/asthma_df1.csv')
>>> dat1
  Phongkham          TenBN
0         A    Tran Van A
1         B  Nguyen Thi B
2         C   Hoang Van C
3         D   Duong Van D
4         E    Pham Thi H
>>> dat3=pd.read_csv('C:/Users/AnhHao/Downloads/pandas/asthma_df3.csv',sep=";")
>>> dat3sub=dat3[6:]
>>> dat3sub
           HoTen Bacsi Gioitinh   FEV1
6     Tran Van A     A      Nam  78.50
7   Nguyen Thi B     B       Nu  84.80
8    Hoang Van C     C      Nam  75.20
9    Duong Van D     D      Nam  74.90
10    Pham Thi H     E       Nu  80.74
>>> m1to1 = pd.merge(left=dat1, right=dat3sub,
... left_on='TenBN', right_on='HoTen')
>>> m1to1
  Phongkham          TenBN          HoTen Bacsi Gioitinh   FEV1
0         A    Tran Van A    Tran Van A     A      Nam  78.50
1         B  Nguyen Thi B  Nguyen Thi B     B       Nu  84.80
2         C   Hoang Van C   Hoang Van C     C      Nam  75.20
3         D   Duong Van D   Duong Van D     D      Nam  74.90
4         E    Pham Thi H    Pham Thi H     E       Nu  80.74
```

Reshaping and Pivoting: Melt function

```
>>> cigarwide=pd.read_csv("C:/Users/AnhHao/Downloads/pandas/cigar_wide.csv")
>>> cigarwide.head(4)
   state  1963  1964  1965  1966  ...   1988   1989   1990   1991   1992
0      1  28.6  29.8  29.8  31.5  ...  114.4  122.3  139.1  144.4  172.2
1      3  23.9  24.0  24.2  29.6  ...  113.5  125.6  130.2  151.4  165.7
2      4  27.0  27.3  27.2  30.3  ...  119.9  127.7  141.2  146.5  177.3
3      5  25.3  25.5  25.3  25.5  ...  117.4  126.4  163.8  186.8  201.9

[4 rows x 31 columns]
>>> cigarlong=pd.melt(cigarwide, id_vars='state', var_name='Year', value_name='Consum')
>>> cigarlong.head(4)
   state  Year  Consum
0      1  1963    28.6
1      3  1963    23.9
2      4  1963    27.0
3      5  1963    25.3
>>> cigarlong.pivot(index='Year',columns='state',values='Consum').head(4)
state    1     3     4     5     7     8   ...    46    47    48    49    50    51
Year                                       ...
1963   28.6  23.9  27.0  25.3  26.8  26.8  ...  28.2  24.7  30.1  27.7  27.6  26.2
1964   29.8  24.0  27.3  25.5  27.8  27.1  ...  29.3  25.2  29.9  28.0  29.5  26.2
1965   29.8  24.2  27.2  25.3  28.1  27.2  ...  29.4  25.1  30.0  28.2  29.8  26.1
1966   31.5  29.6  30.3  25.5  30.1  31.0  ...  32.5  24.7  34.7  28.4  32.1  26.5

[4 rows x 46 columns]
```

Reference: https://notebooks.azure.com/LeNgocKhaNhi/projects

Basic Data Manipulation with SK-Learn

Outline

Feature Scaling with scikit-learn

- Standard Scaler

```
>>> import numpy as np
>>> from sklearn import preprocessing
>>> data = np.array([[1, 4,],
...                  [-5, 8],
...                  [-6, 7],
...                  [9, -5],
...                  [100, -100]])
>>> scaler = preprocessing.StandardScaler()
>>> scaler.fit_transform(data)
array([[-0.46472056,  0.50896876],
       [-0.61303563,  0.6050006 ],
       [-0.63775481,  0.58099264],
       [-0.26696713,  0.29289712],
       [ 1.98247812, -1.98785912]])
```

- Feature is scaled based on: : $x_i' = \frac{x_i - \bar{x}}{\sigma}$ If data is not normally distributed, this is not the best scaler to use.

- Min-Max Scaler

```
>>> minmax=preprocessing.MinMaxScaler()
>>> minmax.fit_transform(data)
array([[0.06603774, 0.96296296],
       [0.00943396, 1.        ],
       [0.        , 0.99074074],
       [0.14150943, 0.87962963],
       [1.        , 0.        ]])
```

- Feature is scaled based on: : $x_i' = \frac{x_i - \min(x)}{\max(x) - min(x)}$ If the distribution is not Gaussian or the standard deviation is very small, the min-max scaler works better

Feature Scaling with scikit-learn

- Robust Scaler

```
>>> robust = preprocessing.RobustScaler()
>>> robust.fit_transform(data)
array([[ 0.        ,  0.        ],
       [-0.42857143,  0.33333333],
       [-0.5       ,  0.25      ],
       [ 0.57142857, -0.75      ],
       [ 7.07142857, -8.66666667]])
```

- Feature is scaled based on: $x_i' = \frac{x_i - Q_1(x)}{Q_1(x) - Q_3(x)}$ suitable for when there are outliers in the data.

- Normlizer

```
>>> normalizer =preprocessing.Normalizer()
>>> normalizer.transform(data)
array([[ 0.24253563,  0.9701425 ],
       [-0.52999894,  0.8479983 ],
       [-0.65079137,  0.7592566 ],
       [ 0.87415728, -0.48564293],
       [ 0.70710678, -0.70710678]])
```

- Feature is scaled based on: $x_i' = \frac{x_i}{\|x\|_2}$ each point is now within 1 unit of the origin on this Cartesian co-ordinate system.

Outline

Handling Categorical Data

- Encoding Nominal Categorical Features

```
>>> from sklearn.preprocessing import LabelBinarizer, MultiLabelBinarizer
>>> multiclass_sign = [("Cough", "Fever"),
... ("Vomit", "Diarrhoea"),
... ("Fever", "Cough"),
... ("Pain", "Diarrhoea")]
>>> sign_multiclass = MultiLabelBinarizer()
>>> sign_multiclass.fit_transform(multiclass_sign)
array([[1, 0, 1, 0, 0],
       [0, 1, 0, 0, 1],
       [1, 0, 1, 0, 0],
       [0, 1, 0, 1, 0]])
>>> sign_multiclass.classes_
array(['Cough', 'Diarrhoea', 'Fever', 'Pain', 'Vomit'], dtype=object)
```

- Encoding Ordinal Categorical Features

```
>>> df = pd.DataFrame({"Score": ["Low", "Medium", "Medium", "High"]})
>>> scale_mapper = {"Low": 1, "Medium": 2, "High": 3}
>>> df["Score"].replace(scale_mapper)
0    1
1    2
2    2
3    3
Name: Score, dtype: int64
```

Encoding Dictionaries of Features

```
>>> from sklearn.feature_extraction import DictVectorizer
>>> sign_dict = [
... {"Pain": 2, "Fever": 4},
... {"Pain": 4, "Fever": 3},
... {"Pain": 1, "Cough": 2},
... {"Pain": 2, "Cough": 2}
... ]
>>> dictvectorizer = DictVectorizer(sparse=False)
>>> dictvectorizer.fit_transform(sign_dict)
array([[0., 4., 2.],
       [0., 3., 4.],
       [2., 0., 1.],
       [2., 0., 2.]])
>>> dictvectorizer.get_feature_names()
['Cough', 'Fever', 'Pain']
```

Read more:

Imputing Missing Class Labels Using k-Nearest Neighbors

Handle Imbalanced Classes In Random Forest

Outline

Cleaning Text

```
>>> text_data = ["  Interrobang. By aishwarya Henriette    ","    Parking And Going. By Karl Gautier"]
>>> strip_whitespace = [string.strip() for string in text_data]
>>> strip_whitespace
['Interrobang. By aishwarya Henriette', 'Parking And Going. By Karl Gautier']
>>> remove_periods = [string.replace(".", "") for string in strip_whitespace]
>>> remove_periods
['Interrobang By aishwarya Henriette', 'Parking And Going By Karl Gautier']
>>> def capitalizer(string: str) -> str:
...     return string.upper()
...
>>> [capitalizer(string) for string in remove_periods]
['INTERROBANG BY AISHWARYA HENRIETTE', 'PARKING AND GOING BY KARL GAUTIER']
>>> import re
>>> def replace_letters_with_X(string: str) -> str:
...     return re.sub(r"[a-zA-Z]", "X", string)
...
>>> [replace_letters_with_X(string) for string in remove_periods]
['XXXXXXXXXXX XX XXXXXXXXX XXXXXXXXX', 'XXXXXXX XXX XXXXX XX XXXX XXXXXXX']
```

- Parsing and Cleaning HTML

```
>>> from bs4 import BeautifulSoup
>>> html = """<div class='full_name'><span style='font-weight:bold'>Yan</span> Chin</div>"""
>>> soup = BeautifulSoup(html)
>>> soup.find("div", {"class": "full_name"}).text
'Yan Chin'
```

Read more: Beautiful Soup documentation

- Removing Punctuation

```
>>> import unicodedata
>>> import sys
>>> text_data = ['Hi!!! I. Love. This. Song.....', '10000% Agree!!!! #LoveIT', 'Right?!?!']
>>> # create a dictionary of punctuation characters
>>> punctuation = dict.fromkeys(i for i in range(sys.maxunicode) if unicodedata.category(chr(i)).startswit
>>> # for each string, remove any punctuation characters
>>> [string.translate(punctuation) for string in text_data]
['Hi I Love This Song', '10000 Agree LoveIT', 'Right']
```

- Tokenizing Text

```
>>> import nltk
>>> string = "The science of today is the technology of tommorw. Tommorrow is today"
>>> word_tokenize = nltk.word_tokenize(string)
>>> print (word_tokenize)
['The', 'science', 'of', 'today', 'is', 'the', 'technology', 'of', 'tommorw', '.', 'Tommorrow', 'is', 'to
>>> sent_tokenize = nltk.sent_tokenize(string)
>>> print (sent_tokenize)
['The science of today is the technology of tommorw.', 'Tommorrow is today']
```

- Removing Stop Words

```
>>> from nltk.corpus import stopwords
>>> tokenized_words = ['i', 'am', 'going', 'to', 'go', 'to', 'the', 'store', 'and', 'park']
>>> stop_words = stopwords.words('english')
>>> [word for word in tokenized_words if word not in stop_words]
['going', 'go', 'store', 'park']
```

Tagging Part of Speech

```
>>> from nltk import pos_tag
>>> from nltk import word_tokenize
>>> import nltk
>>> from sklearn.preprocessing import MultiLabelBinarizer
>>> tweets = ["I am eating a burrito for breakfast",
... "Political science is an amazing field",
... "San Francisco is an awesome city"]
>>> tagged_tweets = []
>>> # tag each word and each tweet
>>> for tweet in tweets:
...     tweet_tag = nltk.pos_tag(word_tokenize(tweet))
...     tagged_tweets.append([tag for word, tag in tweet_tag])
... # use one hot encoding to convert the tags into features
...
>>> one_hot_multi = MultiLabelBinarizer()
>>> one_hot_multi.fit_transform(tagged_tweets)
array([[1, 1, 0, 1, 0, 1, 1, 1, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 1],
       [1, 0, 1, 1, 1, 0, 0, 0, 1]])
>>> one_hot_multi.classes_
array(['DT', 'IN', 'JJ', 'NN', 'NNP', 'PRP', 'VBG', 'VBP', 'VBZ'],
      dtype=object)
```

Encodeing Text as a Bag of Words

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> text_data = np.array(['I love Brazil. Brazil!', 'Sweden is best', 'Germany beats both'])
>>> count = CountVectorizer()
>>> bag_of_words = count.fit_transform(text_data)
>>> bag_of_words.toarray()
array([[0, 0, 0, 2, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 1, 0, 1],
       [1, 0, 1, 0, 1, 0, 0, 0]], dtype=int64)
```

Outline

Handling Dates and Times

- Converting Strings to Dates

```
>>> import numpy as np
>>> import pandas as pd
>>> date_strings = np.array(['03-04-2005 11:35 PM',
... '23-05-2010 12:01 AM',
... '04-09-2009 09:09 PM'])
>>> [pd.to_datetime(date, format='%d-%m-%Y %I:%M %p') for date in date_strings]
[Timestamp('2005-04-03 23:35:00'), Timestamp('2010-05-23 00:01:00'), Timestamp('2009-09-04 21:09:00')]
>>> # Convert to datetimes
>>> [pd.to_datetime(date, format="%d-%m-%Y %I:%M %p", errors="coerce")for date in date_strings]
[Timestamp('2005-04-03 23:35:00'), Timestamp('2010-05-23 00:01:00'), Timestamp('2009-09-04 21:09:00')]
```

- Selecting Dates and Times

```
>>> # Create data frame
>>> dataframe = pd.DataFrame()
>>> # Create datetimes
>>> dataframe['date'] = pd.date_range('1/1/2001', periods=100000, freq='H')
>>> # Select observations between two datetimes
>>> dataframe[(dataframe['date'] > '2002-1-1 01:00:00') & (dataframe['date'] <= '2002-1-1 04:00:00')]
                    date
8762 2002-01-01 02:00:00
8763 2002-01-01 03:00:00
8764 2002-01-01 04:00:00
```

Handling Dates and Times

- Calculating the Difference between Dates

```
>>> # Create data frame
>>> dataframe = pd.DataFrame()
>>> # Create two datetime features
>>> dataframe['Arrived']=[pd.Timestamp('01-01-2017'),pd.Timestamp('01-04-2017')]
>>> dataframe['Left'] = [pd.Timestamp('01-01-2017'), pd.Timestamp('01-06-2017')]
>>> # Calculate duration between features
>>> dataframe['Left'] - dataframe['Arrived']
0   0 days
1   2 days
dtype: timedelta64[ns]
>>> # Calculate duration between features
>>> pd.Series(delta.days for delta in (dataframe['Left'] - dataframe['Arrived']))
0    0
1    2
dtype: int64
```

- Encoding Days of the Week

```
>>> # Create dates
>>> dates = pd.Series(pd.date_range("2/2/2002", periods=3, freq="M"))
>>> # Show days of the week
>>> dates.dt.weekday_name
0    Thursday
1      Sunday
2     Tuesday
dtype: object
```

Outline

Loading Images

```
>>> import cv2
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> fig=plt.figure()
>>> image = cv2.imread("C:/Users/AnhHao/Downloads/plane.jpg", cv2.IMREAD_GRAYSCALE)
>>> plt.imshow(image, cmap="gray"), plt.axis("off")
(<matplotlib.image.AxesImage object at 0x000001F5DB812A20>, (-0.5, 2312.5, 1300.5, -0.5))
>>> fig.savefig('C:/Users/AnhHao/Pictures/myplot.pdf', bbox_inches='tight')
```

Loading Images

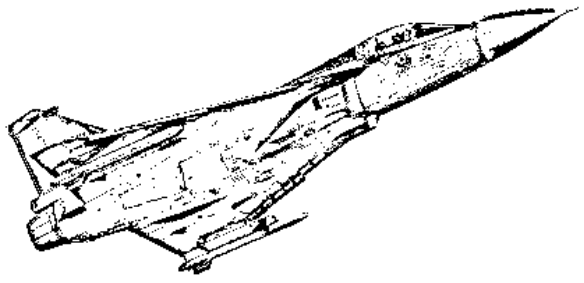
```
>>> # Show image data
>>> image
array([[119, 120, 120, ..., 118, 119, 119],
       [122, 120, 118, ..., 115, 117, 119],
       [119, 117, 118, ..., 116, 118, 117],
       ...,
       [132, 132, 130, ..., 125, 124, 126],
       [131, 131, 132, ..., 130, 124, 125],
       [133, 130, 133, ..., 130, 129, 125]], dtype=uint8)
>>> # Show dimensions
>>> image.shape
(1301, 2313)
>>> # Load image in color
>>> image_bgr = cv2.imread("C:/Users/AnhHao/Downloads/plane.jpg", cv2.IMREAD_COLOR)
>>> image_bgr[0,0]
array([170, 126,  85], dtype=uint8)
>>> image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
>>> plt.imshow(image_rgb), plt.axis("off")
(<matplotlib.image.AxesImage object at 0x000001F5DBAAF3C8>, (-0.5, 2312.5, 1300.5, -0.5))
>>> fig.savefig('C:/Users/AnhHao/Pictures/myplot_rgb.pdf', bbox_inches='tight')
```

Resize Images

```
>>> fig=plt.figure()
>>> # Resize image to 50 pixels by 50 pixels
>>> image_50x50 = cv2.resize(image, (50, 50))
>>> # View image
>>> plt.imshow(image_50x50, cmap="gray"), plt.axis("off")
(<matplotlib.image.AxesImage object at 0x000001F5DDC862B0>, (-0.5, 49.5, 49.5, -0.5))
>>> fig.savefig('C:/Users/AnhHao/Pictures/myplot_50x50.pdf', bbox_inches='tight')
```

Binarizing Images

```
>>> fig=plt.figure()
>>> image_grey = cv2.imread("C:/Users/AnhHao/Downloads/plane.jpg", cv2.IMREAD_GRAYSCALE)
>>> # Apply adaptive thresholding
>>> max_output_value = 255
>>> neighborhood_size = 99
>>> subtract_from_mean = 10
>>> image_binarized = cv2.adaptiveThreshold(image_grey,
...                                         max_output_value,
...                                         cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
...                                         cv2.THRESH_BINARY,
...                                         neighborhood_size,
...                                         subtract_from_mean)
>>> plt.imshow(image_binarized, cmap="gray"), plt.axis("off")
(<matplotlib.image.AxesImage object at 0x000001F5DDCD0780>, (-0.5, 2312.5, 1300.5, -0.5))
>>> fig.savefig('C:/Users/AnhHao/Pictures/myplot_bina.pdf', bbox_inches='tight')
```



Outline

Reducing Features Using Principal Components

- Define: Principal component analysis (PCA) is a statistical procedure that reduce the data dimension without losing information and retain the information needed for building models

Reducing Features Using Principal Components

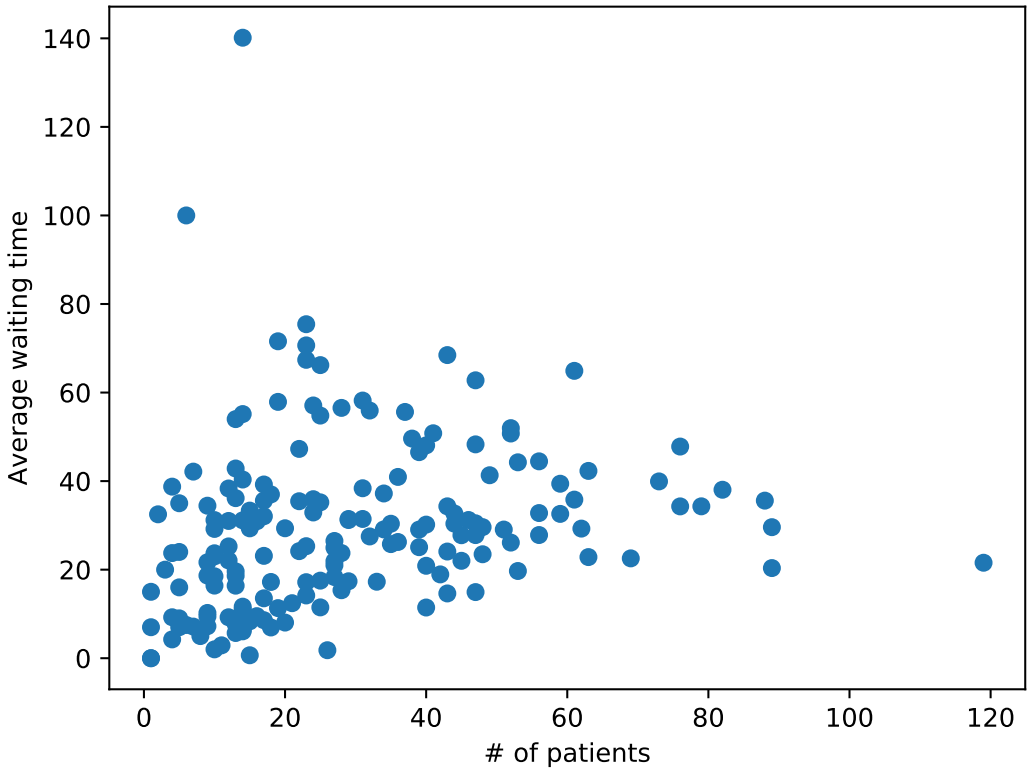
```
>>> # Load libraries
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.decomposition import PCA
>>> from sklearn import datasets

>>> # Load the data
>>> digits = datasets.load_digits()

>>> # Standardize the feature matrix
>>> features = StandardScaler().fit_transform(digits.data)

>>> # Create a PCA that will retain 99% of variance
>>> pca = PCA(n_components=0.99, whiten=True)
>>> # Conduct PCA
>>> features_pca = pca.fit_transform(features)
>>> # Show results
>>> print("Original number of features:", features.shape[1])
Original number of features: 64
>>> print("Reduced number of features:", features_pca.shape[1])
Reduced number of features: 54
```

Practice with a dataset

DCHFS dataset

```
>>> fig=plt.figure()
>>> df=pd.read_csv('D:/anh hao/dchfs/dchfs/datasets/csv/distm11_labeled.csv', na_values=['-2'])
>>> df_cl=df[df['d1126'].notnull()]
>>> df_cl['PATIENT_ID']= df_cl['PATIENT_ID'].astype(str)
>>> def add0(x):
...     return x.rjust(10,'0')
...

>>> # separate patient code and doctor code
>>> paid=df_cl['PATIENT_ID'].apply(add0)
>>> did=paid.str[:8]
>>> pid=paid.str[8:]
>>> df_cl=df_cl.assign(did=did.values,pid=pid.values)
>>> mean=df_cl.groupby('did')['d1126'].mean()
>>> count=df_cl.groupby('did')['pid'].count()
>>> plt.scatter(count, mean)
<matplotlib.collections.PathCollection object at 0x000001F5DE9AF0F0>
>>> plt.xlabel("# of patients")
Text(0.5, 0, '# of patients')
>>> plt.ylabel("Average waiting time")
Text(0, 0.5, 'Average waiting time')
>>> fig.savefig('C:/Users/AnhHao/Pictures/scartter.pdf')
```

DCHFS dataset



DCHFS dataset

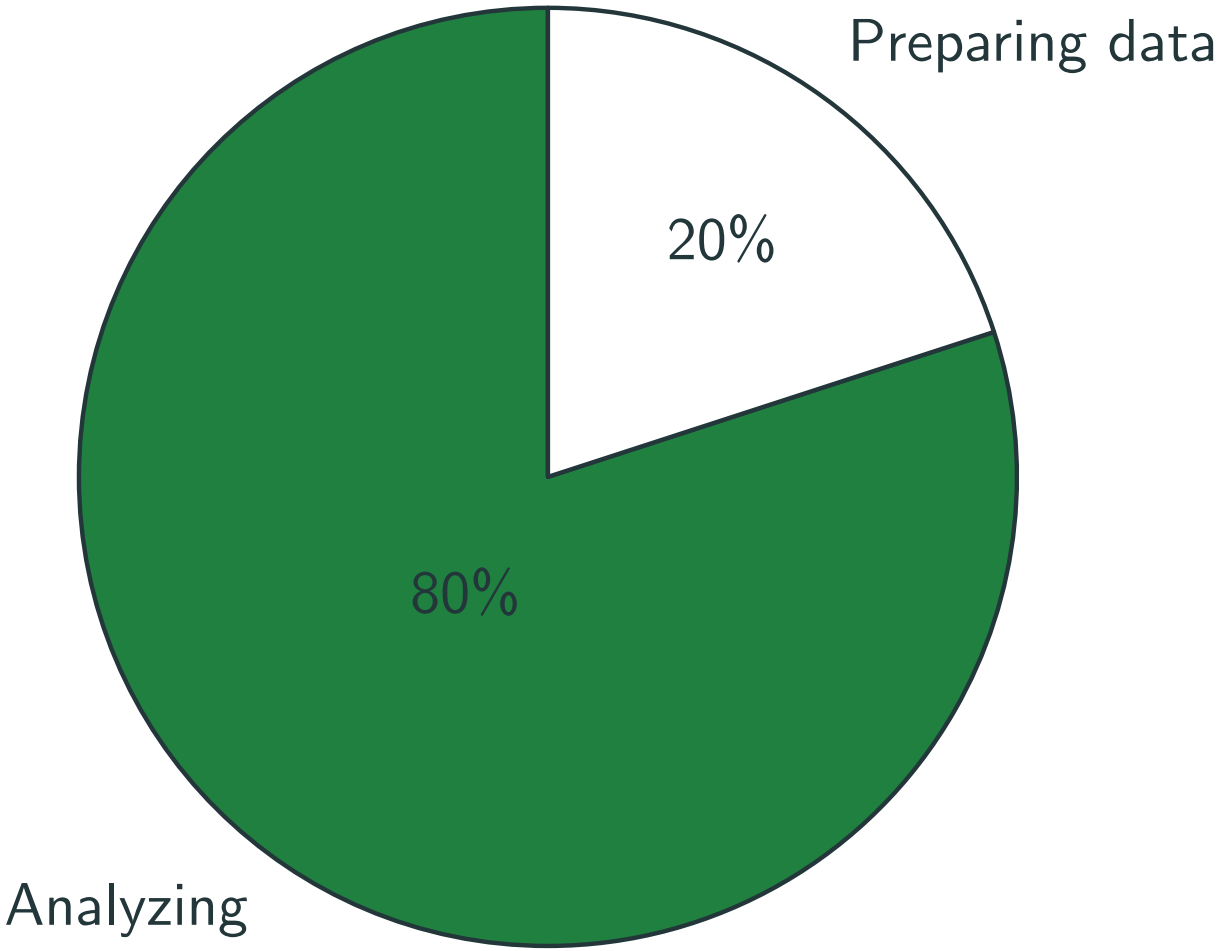
```
>>> dfn=df_cl.loc[:, 'd11351':'d113510']
>>> dfn=dfn.replace(2, 0)
>>> dfn.rename(columns={'d11351': 'Washing machine', 'd11352': 'Water heater', 'd11353': 'Computer',
... 'd11354': 'Refrigerator', 'd11355': 'Gas hob', 'd11356': 'Cell phones',
... 'd11357': 'Cooker','d11358': 'Table', 'd11359': 'Motorcycles', 'd113510': 'TV',}).head(10)
   Washing machine  Water heater  Computer  ...  Table  Motorcycles   TV
0              0.0           0.0       0.0  ...    1.0          1.0  1.0
1              0.0           0.0       0.0  ...    1.0          1.0  1.0
2              0.0           1.0       0.0  ...    1.0          1.0  1.0
3              0.0           0.0       0.0  ...    1.0          0.0  1.0
4              0.0           0.0       0.0  ...    1.0          0.0  1.0
5              0.0           0.0       0.0  ...    0.0          1.0  1.0
6              0.0           0.0       0.0  ...    1.0          1.0  1.0
7              1.0           1.0       0.0  ...    1.0          1.0  1.0
8              0.0           0.0       0.0  ...    1.0          0.0  1.0
9              1.0           1.0       0.0  ...    1.0          1.0  1.0

[10 rows x 10 columns]
```

how to apply pca to calculate the wealth index?

Summary

Summary

- How you'll spend your time ?



Phụ lục

Reference

William McKinney

Python for Data Analysis, 2nd Edition.

O'Reilly Media, 2017.

Joel Grus

Data science from scratch.

O'Reilly Media, 2015.

Chris Albon

Machine Learning with Python Cookbook.

O'Reilly Media, 2018.