https://jakevdp.github.io/PythonDataScienceHandbook/04.00-introduction-to-matplotlib.html

Python Data Science Handbook

导入package

```
import matplotlib.pyplot as plt
plt.style.use('seaborn.whitegrid')
import numpy as np
```

画一个折线图

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('classic')

x=np.linspace(0,10,100)
plt.plot(x,np.sin(x))
plt.plot(x,np.cos(x))
plt.show()
```

选择不同的风格

```
plt.style.use('classic')
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('classic')

x=np.linspace(0,10,100)
plt.plot(x,np.sin(x),'-')
plt.plot(x,np.cos(x),'--')
plt.show() #放在最后，否则可能出问题
```

在IPython里面使用matplotlib

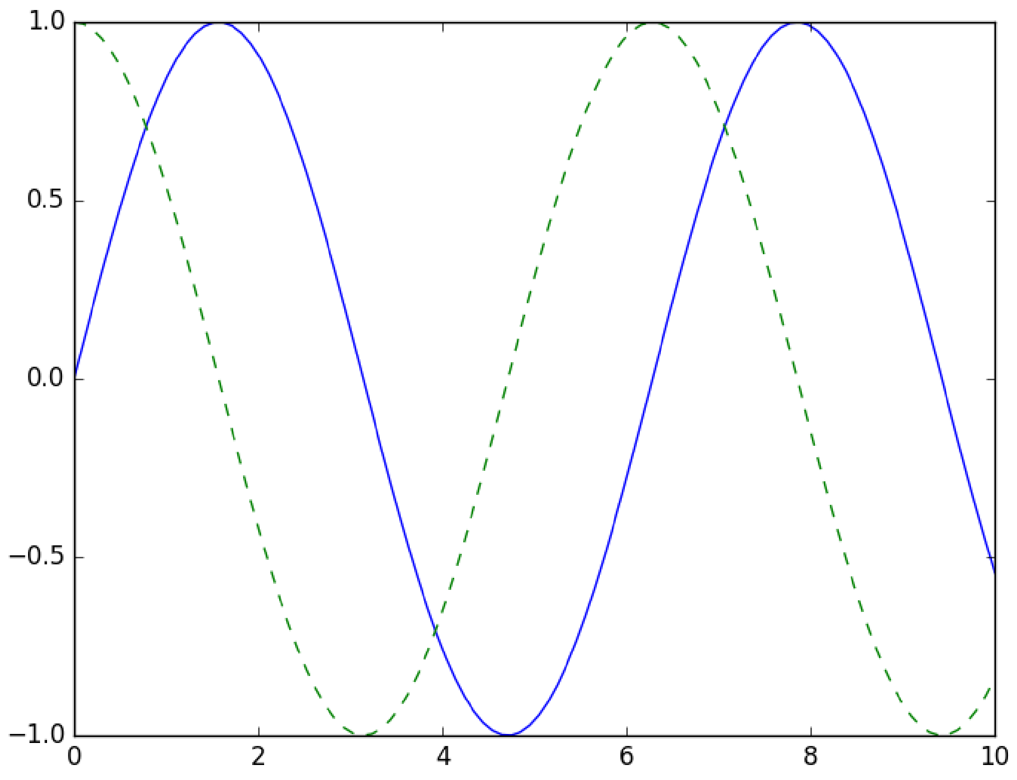
```
%matplotlib
Using matplotlib backend: TkAgg
```

强制更新属性（比如线条的颜色）

```
plt.draw()
```

在notebook里面画图，不需要在使用import matplotlib

```
%matplotlib notebook
%matplotlib inline

import numpy as np
x=np.linspace(0,10,100)
fig=plt.figure()
plt.plot(x,np.sin(x),'-')
plt.plot(x,np.cos(x),'--')
```



```
from Ipython.display import Image
Image('my_figure.png')
```

查看支持的保存的文件格式

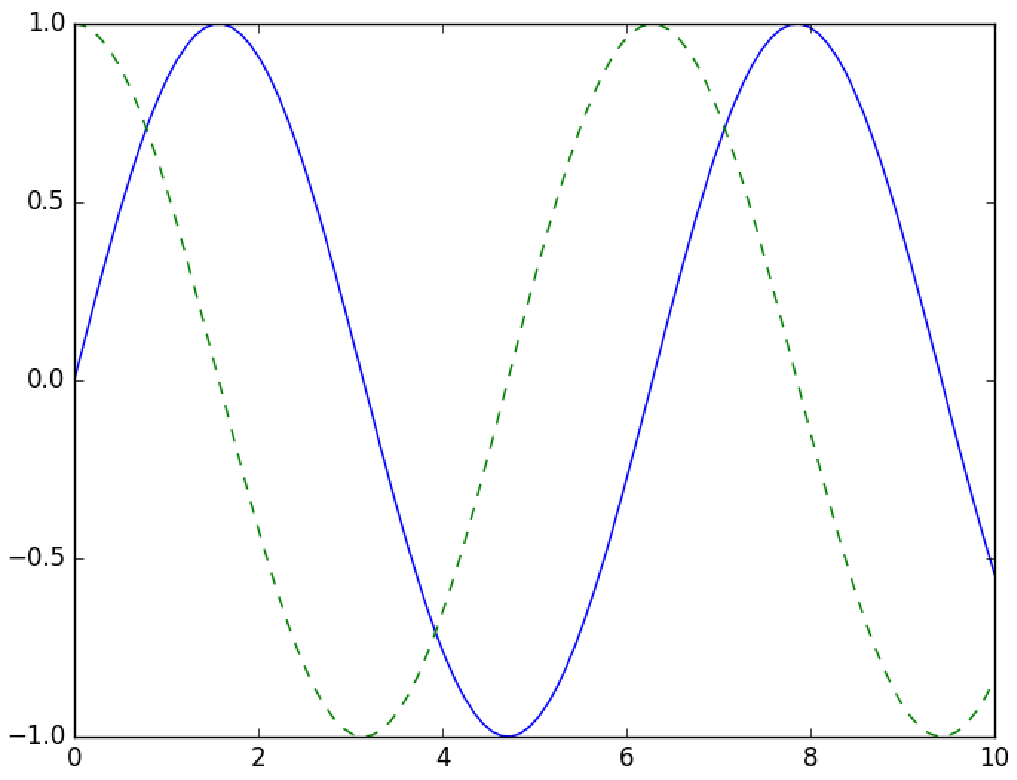
```
print(fig.canvas.get_supported_filetypes())
```

Matlab的风格接口

```
plt.figure()

# 创建上面一行的字图表，并设置x,y轴的数据
plt.subplot(2,1,1)
plt.plot(x,np.sin(x))

# 创建下面第二行的子图表，并设置x,y轴的数据
plt.subplot(2,1,2)
plt.plot(x,np.cos(x))
```

使用plt.gcf()和plt.gca()函数获得函数和维度的引用

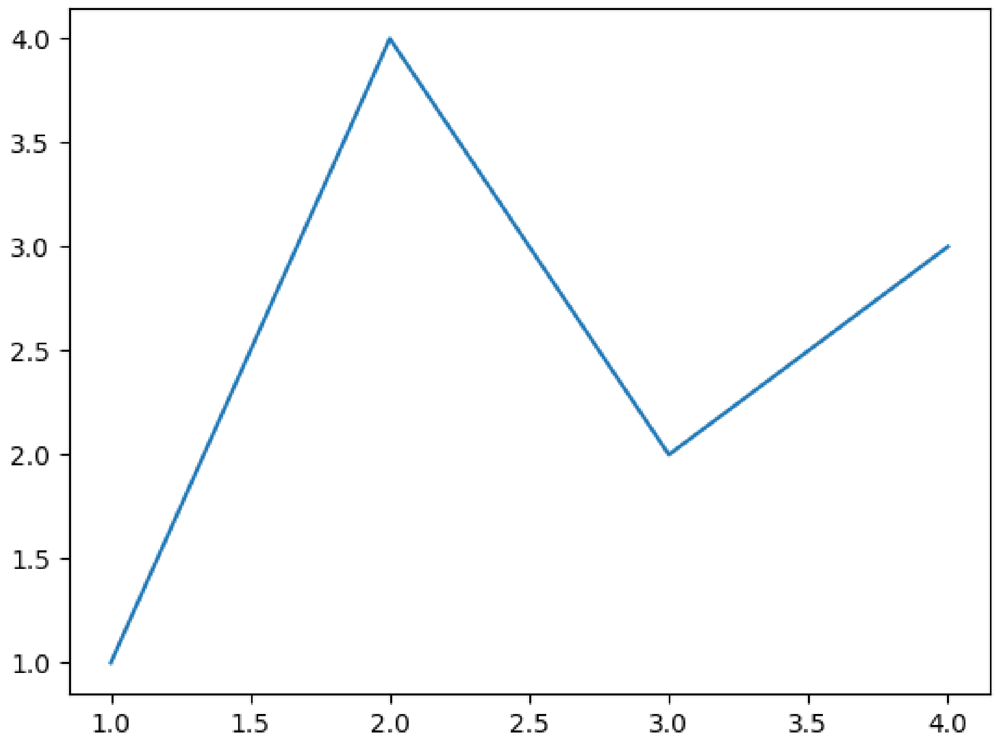
```python
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('classic')

#x=np.linspace(0,10,100)
#fig=plt.figure()
#plt.plot(x,np.sin(x),'-')
#plt.plot(x,np.cos(x),'--')
#plt.show()

#fig.savefig('my_figure.png')
#print(fig.canvas.get_supported_filetypes())

plt.style.use('classic')
fig1=plt.figure()
ax1=plt.axes()

x=np.linspace(0,10,1000)
# 设置color, linestyle
plt.plot(x,np.sin(x-0),color='blue',linestyle='solid')
plt.plot(x,np.sin(x-1),color='g',linestyle='dashed')
plt.plot(x,np.sin(x-2),color='0.75',linestyle='dashdot')
plt.plot(x,np.sin(x-3),color='#FFDD44',linestyle='dotted')
plt.plot(x,np.sin(x-4),color=(1.0,0.2,0.3),linestyle='-.')
plt.plot(x,np.sin(x-5),color='chartreuse',linestyle=':')

# [x_min, x_max,y_min,y_max]
#plt.xlim(-1,11)
#plt.ylim(-1.5,1.5)
plt.axis([-1,11,-1.5,1.5])

# 设置axis与图片的接触距离
plt.axis('tight');
# 设置 equal axis
#plt.axis('equal')


#title, xlabel, ylabel
plt.title("a Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)")

# legend
plt.legend()

#ax=plt.axes()
#ax.plot(x,np.sin(x))
#ax.set(xlim=(0,10),ylim=(-2,2),
#       xlabel='x',ylabel='sin(x)',
#       title='A Simple Plot')


plt.show()
```



参考：https://matplotlib.org/stable/tutorials/text/text_intro.html

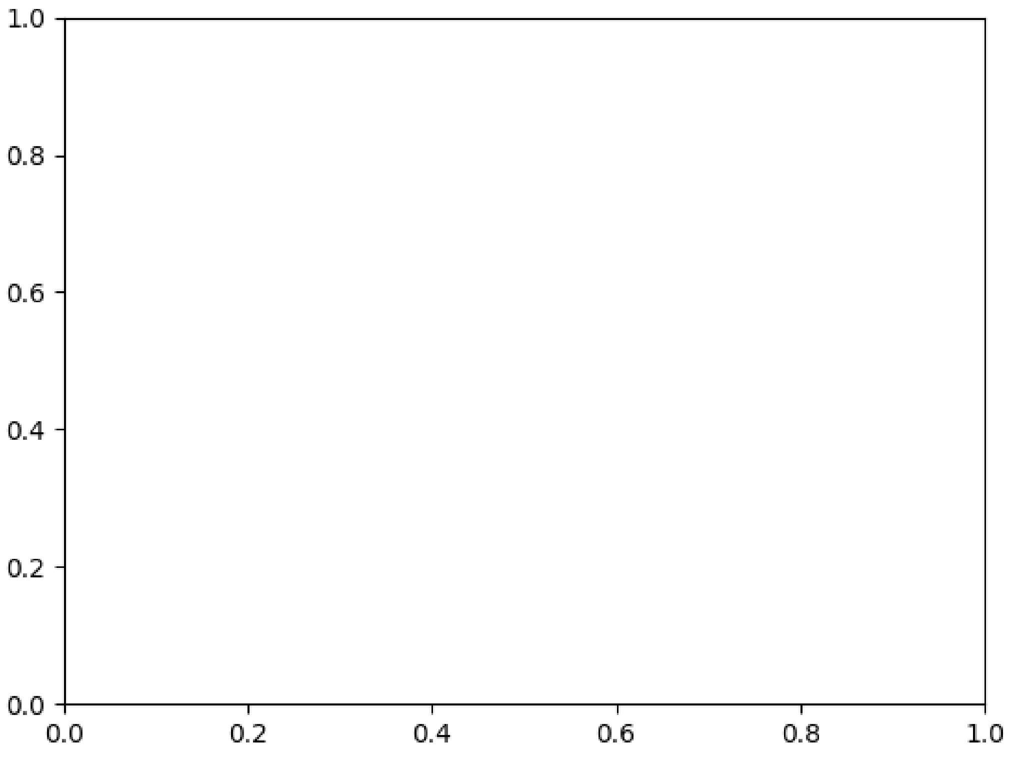
matplotlib画简单的图

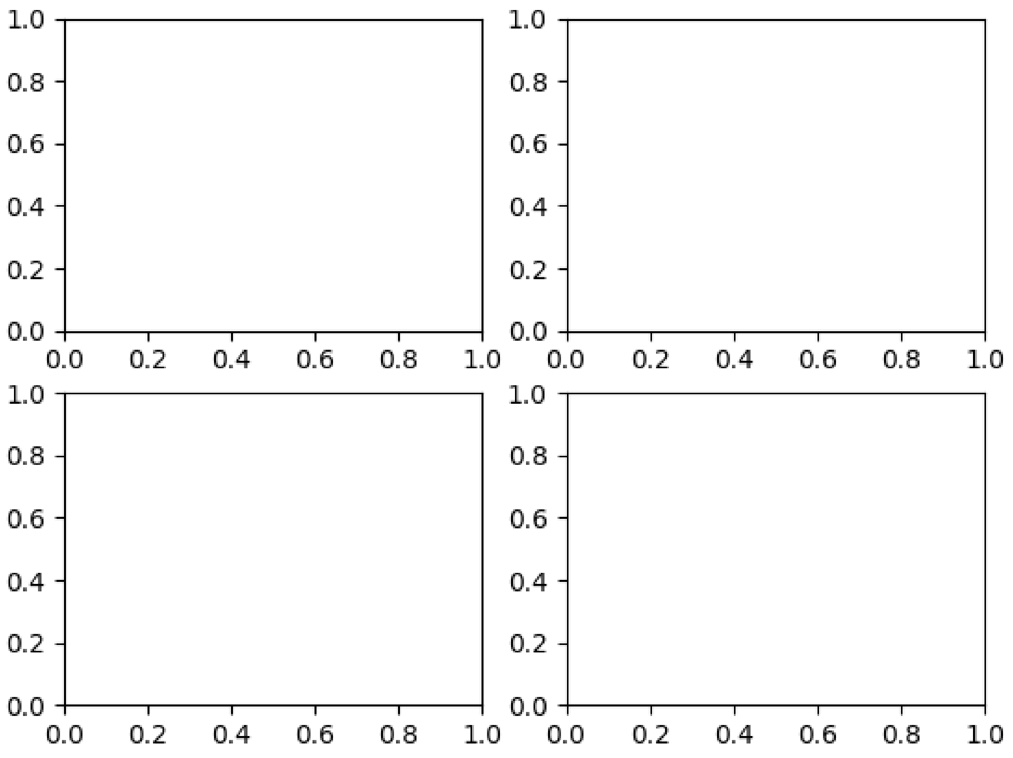
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

fig, ax=plt.subplots()
ax.plot([1,2,3,4],[1,4,2,3]);
plt.show()
```



matplotlib设置简单的图

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

fig1=plt.figure() # an empty figure with no Axes
fig2, ax=plt.subplots() # a figure with a single Axes
fig3, axs =plt.subplots(2,2) # a figure with a 2x2 grid of Axes
plt.show()
fig1.savefig('ex2.png')
fig2.savefig('ex3.png')
fig3.savefig('ex4.png')
```





调整**x,y**轴的范围

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

x=np.linspace(0,10,1000)
fig=plt.plot(x,np.sin(x))
plt.xlim(-1,11)
plt.ylim(-1.5,1.5)
```

输入矩阵

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

b=np.matrix([[1,2],[3,4]])
b_asarray=np.asarray(b)
plt.plot(b_asarray)

plt.show()
```

加入**legend, xlabel,ylabel**

使用**ax**

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)  # Sample data.

# Note that even in the OO-style, we use `.pyplot.figure` to create the Figure.
fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained') # 设置fig, ax
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend();  # Add a legend.
plt.show()
```

使用**plt.plot**来画图

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)  # Sample data.

plt.figure(figsize=(5, 2.7), layout='constrained')
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```

多次重复画图，使用一个helper 函数

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

def my_plotter(ax,data1,data2,param_dict):
    """
    A helper function to make a graph
    """
    out=ax.plot(data1,data2,**param_dict)
    return out

data1,data2,data3,data4=np.random.randn(4,100)
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(5,2.7)) #obtain the fig and ax1, ax2
throught plt.subplots
my_plotter(ax1,data1,data2,{'marker':'x'})
my_plotter(ax2,data3,data4,{'marker':'o'})
plt.show()
```

设置color, linestyle, linewidth

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

data1,data2=np.random.randn(2,100)
fig, ax = plt.subplots(figsize=(5, 2.7))
x = np.arange(len(data1))
ax.plot(x, np.cumsum(data1), color='blue', linewidth=3, linestyle='--')
# use ax.plot
# color:
# line width:
# linestyle
l, = ax.plot(x, np.cumsum(data2), color='orange', linewidth=2)
# use handle=ax.plot
#  use handle.set_linestyle()
l.set_linestyle(':')
plt.show()
```

设置marker,facecolor

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

data1,data2=np.random.randn(2,100)
fig, ax = plt.subplots(figsize=(5, 2.7))
ax.scatter(data1, data2, s=50, facecolor='C0', edgecolor='k')
# facecolor: background color
# edgecolor: color of marker edge
plt.show()
```

设置linewidth, linestyles, markersize

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

data1,data2,data3,data4=np.random.randn(4,100)
fig, ax = plt.subplots(figsize=(5, 2.7))
ax.plot(data1, marker='o', label='data1',markersize=4)
#  marker: type of marker
#  label: legend
# markersize: 4
ax.plot(data2, 'd', label='data2')
ax.plot(data3, 'v', label='data3')
ax.plot(data4, 's', label='data4')
ax.legend()
plt.show()
```

设置坐标轴的文字和说明文字

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
mu, sigma = 115, 15
x = mu + sigma * np.random.randn(10000)
fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')
# the histogram of the data
n, bins, patches = ax.hist(x, 50, density=True, facecolor='C0', alpha=0.75)

ax.set_xlabel('Length [cm]',fontsize=14,color='red')
# 设置fontsize, color
ax.set_ylabel('Probability')
ax.set_title('Aardvark lengths\n (not really)')
# 使用\n来换行
ax.text(75, .025, r'$\mu=115,\ \sigma=15$')
# 使用ax.text来输入文字
ax.axis([55, 175, 0, 0.03])
ax.grid(True)
# 画grid
plt.show()
```

设置注释文字和箭头

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(figsize=(5, 2.7))

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2 * np.pi * t)
line, = ax.plot(t, s, lw=2)

ax.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
            arrowprops=dict(facecolor='blue', shrink=0.05))
# ax.annotate是用来注释
# xy:  箭头终点的位置
# xytext: 文字的位置
# facecolor:箭头的颜色
# shrink箭头到文字的距离
ax.set_ylim(-2, 2)
plt.show()
```

设置坐标轴尺度

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
data1=np.random.randn(1,100)
fig, axs = plt.subplots(1, 2, figsize=(5, 2.7), layout='constrained')
xdata = np.arange(len(data1))  # make an ordinal for this
data = 10**data1
axs[0].plot(xdata, data)

axs[1].set_yscale('log')
# axs[1]设置
# set_yscale():设置y坐标尺度
# set_xscale():设置x坐标尺度
# 'log'

axs[1].plot(xdata, data);
plt.show()
```

设置**tick**的值

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
xdata=np.random.randn(1,100)
data1=np.random.randn(1,len(xdata))
fig, axs = plt.subplots(2, 1, layout='constrained')
axs[0].plot(xdata, data1)
axs[0].set_title('Automatic ticks')

axs[1].plot(xdata, data1)
axs[1].set_xticks(np.arange(0, 100, 30), ['zero', '30', 'sixty', '90'])
# ['zero','30','sixty','90']是xtick的值
axs[1].set_yticks([-1.5, 0, 1.5])  # note that we don't need to specify labels
```

```
axs[1].set_title('Manual ticks');
plt.show()
```

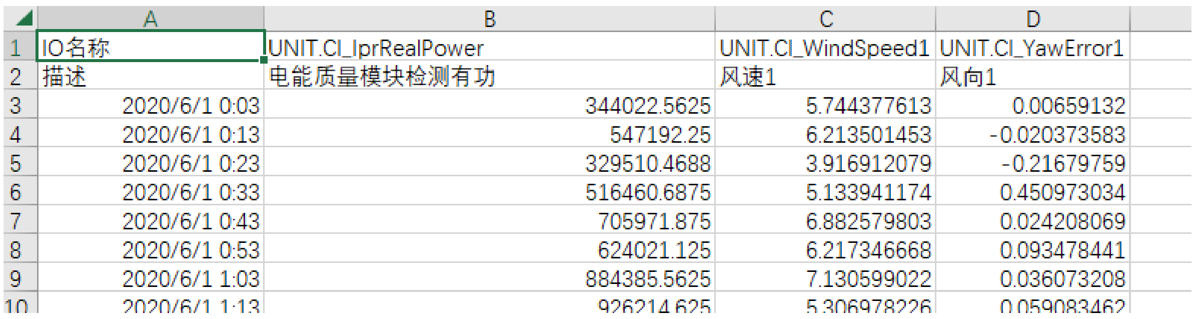
保存为矢量图

```
savefig('file_name',dpi=400,format='svg',bbox_inches='tight')
savefig('file_name',dpi=400,format='svg',bbox_inches='tight')
```

https://blog.csdn.net/sinat_39620217/article/details/119924912?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-0-119924912-blog-79640467.pc_relevant_3mothn_strategy_recovery&spm=1001.2101.3001.4242.1&utm_relevant_index=2

pandas package的具体用法

风力功率文件说明



	A	B	C	D
1	IO名称	UNIT.CI_IprRealPower	UNIT.CI_WindSpeed1	UNIT.CI_YawError1
2	描述	电能质量模块检测有功	风速1	风向1
3	2020/6/1 0:03	344022.5625	5.744377613	0.00659132
4	2020/6/1 0:13	547192.25	6.213501453	-0.020373583
5	2020/6/1 0:23	329510.4688	3.916912079	-0.21679759
6	2020/6/1 0:33	516460.6875	5.133941174	0.450973034
7	2020/6/1 0:43	705971.875	6.882579803	0.024208069
8	2020/6/1 0:53	624021.125	6.217346668	0.093478441
9	2020/6/1 1:03	884385.5625	7.130599022	0.036073208
10	2020/6/1 1:13	926214.625	5.306978226	0.059083462

自动导入参数设置

```
## filename: run.py

import argparse

# sete the parameter's description
parser = argparse.ArgumentParser(description='Non-stationary Transformers for
Time Series Forecasting')

# 'parameter', type= type of parameter, default: default value, help: help
information
# parser.add_argument: add one argument:
#
# basic config: for different parts
```

```
parser.add_argument('--is_training', type=int, required=True, default=1,
help='status')
parser.add_argument('--model_id', type=str, required=True, default='test',
help='model id')
parser.add_argument('--model', type=str, required=True, default='Transformer',
                    help='model name, options: [ns_Transformer, Transformer]')

# parse.add_argument('name', \
#                      "-a","--all" fulname
#                      default= 'default value',\
#                      required=True (we can omit this),\
#                      type=str,\
#                      help="help information",\
#                      dest= The name of the attribute to be added,\
#                      metavar: for the usage message (nickname),\
#                       nargs='+' (means: arbitrary parameters),number of
arugments for this parameter
#                        const:
#                        choices: choice=['a','b','c'], then we can only chooise
from this container
#                        action: basic actions type: such as store_true
# python prog.py -h: type the help information
# parser.print_help()

# Now, we can use the arguments
args = parser.parse_args()
```

自动运行多组超参数实验

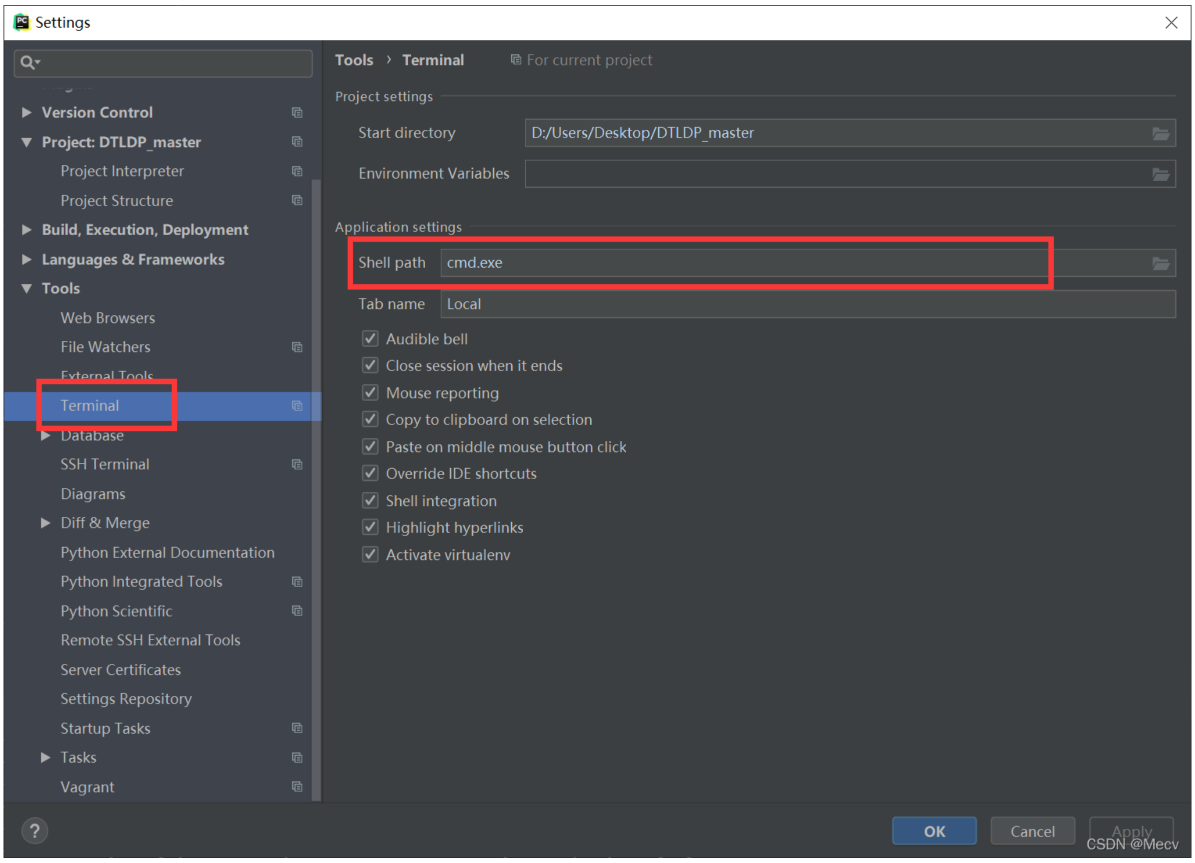
```
# file_name: ns_informer.sh
# use (in ： bash ns_informer.sh

# set the parameters 1
# is_training is parameter; 1 is argument
# root_path: the path of data set
# data_path: the name of dataset
# model_id:
# model: the name of models
# data:
# features:
# gpu: number of gpus
# des: destination of result?
python -u run.py \
  --is_training 1 \
  --root_path ./dataset/electricity/ \
  --data_path electricity.csv \
  --model_id ECL_96_96 \
  --model ns_Informer \
  --data custom \
  --features M \
  --seq_len 96 \
  --label_len 48 \
  --pred_len 96 \
  --e_layers 2 \
  --d_layers 1 \
  --factor 3 \
  --enc_in 321 \
  --dec_in 321 \
  --c_out 321 \
  --gpu 0 \
  --des 'Exp_h256_l2' \
  --p_hidden_dims 256 256 \
  --p_hidden_layers 2 \
  --itr 1 &


python -u run.py \
  --is_training 1 \
  --root_path ./dataset/electricity/ \
  --data_path electricity.csv \
  --model_id ECL_96_192 \
  --model ns_Informer \
  --data custom \
  --features M \
  --seq_len 96 \
  --label_len 48 \
  --pred_len 192 \
  --e_layers 2 \
  --d_layers 1 \
  --factor 3 \
  --enc_in 321 \
  --dec_in 321 \
  --c_out 321 \
  --gpu 1 \
  --des 'Exp_h256_l2' \
  --p_hidden_dims 256 256 \
  --p_hidden_layers 2 \
  --itr 1 &
```

安装 运行**sh**文件所需要的**Git**软件

参考博客：csdn博客

1，下载Git软件

2，打开pycharm的如图所示的界面


Settings

Tools › Terminal   For current project

Project settings

Start directory: D:/Users/Desktop/DTLDP_master

Environment Variables:

Application settings

Shell path: cmd.exe

Tab name: Local

- ✓ Audible bell
- ✓ Close session when it ends
- ✓ Mouse reporting
- ✓ Copy to clipboard on selection
- ✓ Paste on middle mouse button click
- ✓ Override IDE shortcuts
- ✓ Shell integration
- ✓ Highlight hyperlinks
- ✓ Activate virtualenv

3， 把shell path换成Git安装目录下bin文件夹里面的sh.exe

自动安装所需要的环境

```
# file_name: requirements.txt
# run: pip install -r requirements.txt
pandas
sklearn
torchvision
numpy
matplotlib
torch
```

安装yml所列出的环境

```
# file_name: environment.yml
name: nsformer
channels:
  - conda-forge
  - pytorch
  - defaults
dependencies:
  - python=3.7
  - pip
  - matplotlib
  - numpy
  - pandas
  - scikit-learn
  - pip:
    - torch==1.9.0
```

use the following cmd

```
conda env update --file environment.yml
```

开源模板

一些常用的小tips

查询函数被哪些文件使用过

find usage in right click

把常用函数都写成一个modulus

放在同一个

写自己的Class

安装对应于GPU的pytorch版本

测试是否有GPU所需的cuda

```
import torch
print(torch.cuda.is_available())
```

查询驱动的版本

```
# in cmd:  查看Driver API
nvidia-smi
# in power shell: 查看runtime API
nvcc -V
# Driver Version 527.56
```

查询对应的cuda版本:

Nvidia control panel----》 system information----->NVCuda64 DLL, 12.0.94 driver

安装cuda，去掉Visual studio integration, 和Nsight的勾选

一个重要的说明

使用conda安装的时候，由于package: pytorch-mutex的存在，会在找不到GPU版本的时候找一个CPU版本安装，所以会导致conda安装的pytorch版本是CPU版本的。

解决办法: 本地安装package,不使用在线的版本，参考链接知乎文章

进入链接： https://download.pytorch.org/whl/torch_stable.html

选择对应的版本：

[cu117/torch-1.13.0%2Bcu117-cp39-cp39-win_amd64.whl]

cu117: cuda 11.7

cp39: python version 3.9

win amd64

下载 torch, torchvision, torchaudio(先安装torch,再安装torchvision， 否者pip 会自己再下一个torch安装过去), 这里需要注意版本，并不是所有的python==3.9.2这种版本都可以的，其中python==3.9.13这个版本是可以的

```
D:
cd D:\WPF\local_package
pip install torch-1.13.0+cu117-cp39-cp39-win_amd64.whl
pip install torchaudio-0.13.0+cu117-cp39-cp39-win_amd64.whl
pip install torchvision-0.14.0+cu117-cp39-cp39-win_amd64.whl
```

测试是否安装成功

```
python
import torch
torch.cuda.is_available()
>True
torch.__version__
>'1.13.0+cu117'
print(torch.version.cuda)
>11.7
```

在pycharm里面替换某一个单词

```
Ctrl+R #替换
```

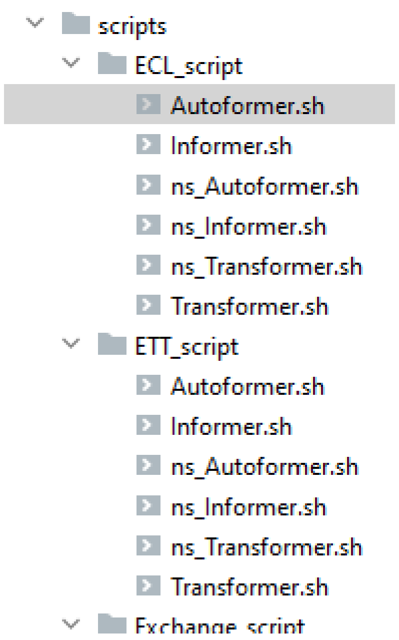
pycharm打开项目文件

```
File--->Open
```

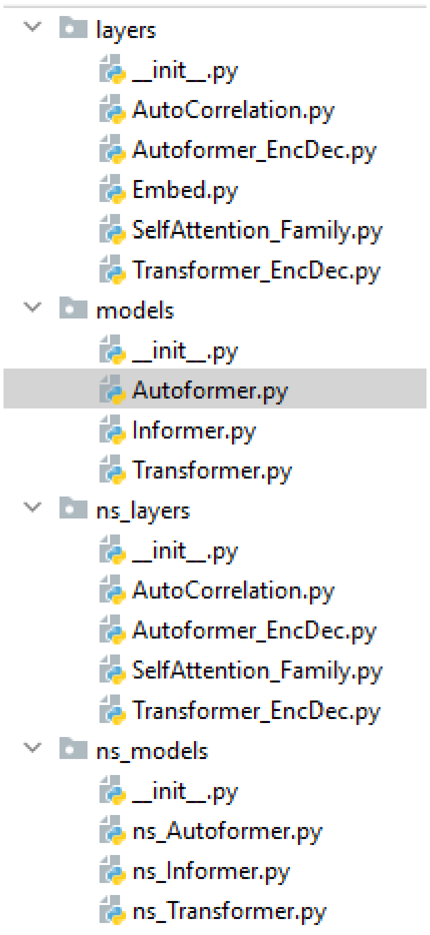
安装sklearn

```
pip install scikit-learn
```

针对多个时序数据集提出统一的算法


scripts
ECL_script
Autoformer.sh
Informer.sh
ns_Autoformer.sh
ns_Informer.sh
ns_Transformer.sh
Transformer.sh
ETT_script
Autoformer.sh
Informer.sh
ns_Autoformer.sh
ns_Informer.sh
ns_Transformer.sh
Transformer.sh
Exchange_script

构造网络的文件框架：**layer(folder), model(folder)**


layers
__init__.py
AutoCorrelation.py
Autoformer_EncDec.py
Embed.py
SelfAttention_Family.py
Transformer_EncDec.py
models
__init__.py
Autoformer.py
Informer.py
Transformer.py
ns_layers
__init__.py
AutoCorrelation.py
Autoformer_EncDec.py
SelfAttention_Family.py
Transformer_EncDec.py
ns_models
__init__.py
ns_Autoformer.py
ns_Informer.py
ns_Transformer.py

如何查阅特定函数和类的文档

```
import torch
# 查询随机数生成模块中的所有属性
## __funtion__: 特殊对象
## _function_: 内部函数
print(dir(torch.distributions))

# 查询特定的函数和类的用法
help(torch.ones)
```

```
# 使用这个函数
torch.ones(4)


# 在jupyter notebook
list? #在新的记事本中显示
list?? #显示实现的代码
```

代码文件夹的结构

data_provider

exp

figures

layers

models

script

utils

run.py

requirement.txt

实验室Titan显卡的账号信息

识别码：264550727

验证码：8YnK2L

一些常见的Error

RuntimeError on windows trying python multiprocessing

```
RuntimeError:
            Attempt to start a new process before the current process
            has finished its bootstrapping phase.
            This probably means that you are on Windows and you have
            forgotten to use the proper idiom in the main module:
                if __name__ == '__main__':
                    freeze_support()
                    ...
            The "freeze_support()" line can be omitted if the program
            is not going to be frozen to produce a Windows executable.
```

解决方案：StackOverflow

在原本的main函数里面加入

```
if __name__='__main__'
```

testMain.py

```
import parallelTestModule

extractor = parallelTestModule.ParallelExtractor()
extractor.runInParallel(numProcesses=2, numThreads=4)
```

变为

```
import parallelTestModule

if __name__ == '__main__':
    extractor = parallelTestModule.ParallelExtractor()
    extractor.runInParallel(numProcesses=2, numThreads=4)
```

常用模块

是否有GPU

```
import argparse
parser = argparse.ArgumentParser(description='Non-stationary Transformers for
Time Series Forecasting')

# GPU
parser.add_argument('--use_gpu', type=bool, default=True, help='use gpu')
parser.add_argument('--gpu', type=int, default=0, help='gpu')
parser.add_argument('--use_multi_gpu', action='store_true', help='use multiple
gpus', default=False)
parser.add_argument('--devices', type=str, default='0,1,2,3', help='device ids
of multile gpus')
parser.add_argument('--seed', type=int, default=2021, help='random seed')
args = parser.parse_args()
args.use_gpu = True if torch.cuda.is_available() and args.use_gpu else False
```

设定随机数生成种子保证复现

```
fix_seed = args.seed
random.seed(fix_seed)
torch.manual_seed(fix_seed)
np.random.seed(fix_seed)
```

检查是否有多个GPU

```
if args.use_gpu:
    if args.use_multi_gpu:
        args.devices = args.devices.replace(' ', '')
        device_ids = args.devices.split(',')
        args.device_ids = [int(id_) for id_ in device_ids]
        args.gpu = args.device_ids[0]
    else:
        torch.cuda.set_device(args.gpu)
```

打印实验参数设置

```
print('Args in experiment:')
print(args)
```

生成实验参数**String,**进行实验

```
Exp = Exp_Main

if args.is_training:
    for ii in range(args.itr):
        # setting record of experiments
        setting = 
'{}_{}_{}_ft{}_sl{}_ll{}_pl{}_dm{}_nh{}_el{}_dl{}_df{}_fc{}_eb{}_dt{}_{}_{}'.for
mat(
            args.model_id,
            args.model,
            args.data,
            args.features,
            args.seq_len,
            args.label_len,
            args.pred_len,
            args.d_model,
            args.n_heads,
            args.e_layers,
            args.d_layers,
            args.d_ff,
            args.factor,
            args.embed,
            args.distil,
            args.des, ii)

        exp = Exp(args)  # set experiments
        print('>>>>>>>start training : 
{}>>>>>>>>>>>>>>>>>>>>>>>>>>'.format(setting))
        exp.train(setting)

        print('>>>>>>>testing : {} 
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<'.format(setting))
        exp.test(setting)

        if args.do_predict:
            print('>>>>>>>predicting : {} 
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<'.format(setting))
            exp.predict(setting, True)
        torch.cuda.empty_cache()
else:
    ii = 0
    setting = 
'{}_{}_{}_ft{}_sl{}_ll{}_pl{}_dm{}_nh{}_el{}_dl{}_df{}_fc{}_eb{}_dt{}_{}_{}'.for
mat(args.model_id,

                                                                  args.model,

                                                                  args.data,

                                                                  args.features,

                                                                  args.seq_len,

                                                                  args.label_len,

                                                                  args.pred_len,

                                                                  args.d_model,

                                                                  args.n_heads,

                                                                  args.e_layers,

                                                                  args.d_layers,

                                                                  args.d_ff,

                                                                  args.factor,

                                                                  args.embed,

                                                                  args.distil,

                                                                  args.des, ii)

    exp = Exp(args)  # set experiments
    print('>>>>>>>testing : {}
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<'.format(setting))
    exp.test(setting, test=1)
    torch.cuda.empty_cache()
```

数据 data_provider

实验部分代码

```
# Exp_Main是一个class
Exp = Exp_Main

if args.is_training:
    for ii in range(args.itr):
        # setting record of experiments
        setting =
'{}_{}_{}_ft{}_sl{}_ll{}_pl{}_dm{}_nh{}_el{}_dl{}_df{}_fc{}_eb{}_dt{}_{}_{}'.for
mat(
            args.model_id,
            args.model,
            args.data,
            args.features,
            args.seq_len,
            args.label_len,
            args.pred_len,
            args.d_model,
            args.n_heads,
            args.e_layers,
            args.d_layers,
            args.d_ff,
            args.factor,
            args.embed,
            args.distil,
            args.des, ii)

        exp = Exp(args)  # set experiments
        print('>>>>>>>start training :
{}>>>>>>>>>>>>>>>>>>>>>>>>>>>'.format(setting))
        exp.train(setting)

        print('>>>>>>>testing : {}
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<'.format(setting))
        exp.test(setting)

        if args.do_predict:
            print('>>>>>>>predicting : {}
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<'.format(setting))
            exp.predict(setting, True)

        torch.cuda.empty_cache()
else:
    ii = 0
    setting =
'{}_{}_{}_ft{}_sl{}_ll{}_pl{}_dm{}_nh{}_el{}_dl{}_df{}_fc{}_eb{}_dt{}_{}_{}'.for
mat(args.model_id,

                 args.model,

                 args.data,

                 args.features,

                 args.seq_len,

                 args.label_len,

                 args.pred_len,

                 args.d_model,

                 args.n_heads,

                 args.e_layers,
                 args.d_layers,

                 args.d_ff,

                 args.factor,

                 args.embed,

                 args.distil,

                 args.des, ii)

    exp = Exp(args)  # set experiments
    print('>>>>>>>testing : {}
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<'.format(setting))
    exp.test(setting, test=1)
    torch.cuda.empty_cache()
```

```
from data_provider.data_factory import data_provider
from exp.exp_basic import Exp_Basic
# 导入之前的模型
from models import Transformer, Informer, Autoformer
# 导入自己的模型
from ns_models import ns_Transformer, ns_Informer, ns_Autoformer
from utils.tools import EarlyStopping, adjust_learning_rate, visual
# 导入度量函数，性能指标
from utils.metrics import metric

import numpy as np
import torch
import torch.nn as nn
from torch import optim

import os
import time

import warnings
import matplotlib.pyplot as plt
import numpy as np
class Exp_Main(Exp_Basic):
    def __init__(self, args):
        super(Exp_Main, self).__init__(args)

    def _build_model(self):
        model_dict = {
            'Transformer': Transformer,
            'Informer': Informer,
            'Autoformer': Autoformer,
            'ns_Transformer': ns_Transformer,
            'ns_Informer': ns_Informer,
            'ns_Autoformer': ns_Autoformer,
        }
        model = model_dict[self.args.model].Model(self.args).float()

        if self.args.use_multi_gpu and self.args.use_gpu:
            model = nn.DataParallel(model, device_ids=self.args.device_ids)
        return model

    def _get_data(self, flag):
        data_set, data_loader = data_provider(self.args, flag)
        return data_set, data_loader

    def _select_optimizer(self):
        model_optim = optim.Adam(self.model.parameters(),
lr=self.args.learning_rate)
        return model_optim

    def _select_criterion(self):
        criterion = nn.MSELoss()
        return criterion

    def vali(self, vali_data, vali_loader, criterion):
        total_loss = []
        self.model.eval()
        with torch.no_grad():
            for i, (batch_x, batch_y, batch_x_mark, batch_y_mark) in
enumerate(vali_loader):
                batch_x = batch_x.float().to(self.device)
                batch_y = batch_y.float()

                batch_x_mark = batch_x_mark.float().to(self.device)
                batch_y_mark = batch_y_mark.float().to(self.device)

                # decoder input
                dec_inp = torch.zeros_like(batch_y[:, -self.args.pred_len:,
:]).float()
                dec_inp = torch.cat([batch_y[:, :self.args.label_len, :],
dec_inp], dim=1).float().to(self.device)
                # encoder - decoder
                if self.args.use_amp:
                    with torch.cuda.amp.autocast():
                        if self.args.output_attention:
                            outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)[0]
                        else:
                            outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)
                else:
                    if self.args.output_attention:
                        outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)[0]
                    else:
                        outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)
                f_dim = -1 if self.args.features == 'MS' else 0
                outputs = outputs[:, -self.args.pred_len:, f_dim:]
                batch_y = batch_y[:, -self.args.pred_len:,
f_dim:].to(self.device)

                pred = outputs.detach().cpu()
                true = batch_y.detach().cpu()

                loss = criterion(pred, true)
                total_loss.append(loss)
        total_loss = np.average(total_loss)
        self.model.train()
        return total_loss

    def train(self, setting):
        train_data, train_loader = self._get_data(flag='train')
        vali_data, vali_loader = self._get_data(flag='val')
        test_data, test_loader = self._get_data(flag='test')

        path = os.path.join(self.args.checkpoints, setting)
        if not os.path.exists(path):
            os.makedirs(path)

        time_now = time.time()

        train_steps = len(train_loader)
        early_stopping = EarlyStopping(patience=self.args.patience,
verbose=True)

        model_optim = self._select_optimizer()
        criterion = self._select_criterion()

        if self.args.use_amp:
            scaler = torch.cuda.amp.GradScaler()

        for epoch in range(self.args.train_epochs):
            iter_count = 0
            train_loss = []

            self.model.train()
            epoch_time = time.time()
            for i, (batch_x, batch_y, batch_x_mark, batch_y_mark) in
enumerate(train_loader):
                iter_count += 1
                model_optim.zero_grad()
                batch_x = batch_x.float().to(self.device)

                batch_y = batch_y.float().to(self.device)
                batch_x_mark = batch_x_mark.float().to(self.device)
                batch_y_mark = batch_y_mark.float().to(self.device)

                # decoder input
                dec_inp = torch.zeros_like(batch_y[:, -self.args.pred_len:,
:]).float()
                dec_inp = torch.cat([batch_y[:, :self.args.label_len, :],
dec_inp], dim=1).float().to(self.device)

                # encoder - decoder
                if self.args.use_amp:
                    with torch.cuda.amp.autocast():
                        if self.args.output_attention:
                            outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)[0]
                        else:
                            outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)
```

```
                        f_dim = -1 if self.args.features == 'MS' else 0
                        outputs = outputs[:, -self.args.pred_len:, f_dim:]
                        batch_y = batch_y[:, -self.args.pred_len:,
f_dim:].to(self.device)
                        loss = criterion(outputs, batch_y)
                        train_loss.append(loss.item())
                else:
                    if self.args.output_attention:
                        outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)[0]
                    else:
                        outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark, batch_y)

                    f_dim = -1 if self.args.features == 'MS' else 0
                    outputs = outputs[:, -self.args.pred_len:, f_dim:]
                    batch_y = batch_y[:, -self.args.pred_len:,
f_dim:].to(self.device)
                    loss = criterion(outputs, batch_y)
                    train_loss.append(loss.item())

                if (i + 1) % 100 == 0:
                    print("\titers: {0}, epoch: {1} | loss: {2:.7f}".format(i +
1, epoch + 1, loss.item()))
                    speed = (time.time() - time_now) / iter_count
                    left_time = speed * ((self.args.train_epochs - epoch) *
train_steps - i)
                    print('\tspeed: {:.4f}s/iter; left time:
{:.4f}s'.format(speed, left_time))
                    iter_count = 0
                    time_now = time.time()

                if self.args.use_amp:
                    scaler.scale(loss).backward()
                    scaler.step(model_optim)
                    scaler.update()
                else:
                    loss.backward()
                    model_optim.step()

            print("Epoch: {} cost time: {}".format(epoch + 1, time.time() -
epoch_time))
            train_loss = np.average(train_loss)
            vali_loss = self.vali(vali_data, vali_loader, criterion)
            test_loss = self.vali(test_data, test_loader, criterion)

            print("Epoch: {0}, Steps: {1} | Train Loss: {2:.7f} Vali Loss:
{3:.7f} Test Loss: {4:.7f}".format(
                epoch + 1, train_steps, train_loss, vali_loss, test_loss))
            early_stopping(vali_loss, self.model, path)
            if early_stopping.early_stop:
                print("Early stopping")
                break

            adjust_learning_rate(model_optim, epoch + 1, self.args)

        best_model_path = path + '/' + 'checkpoint.pth'
        self.model.load_state_dict(torch.load(best_model_path))
        return self.model

    def test(self, setting, test=0):
        test_data, test_loader = self._get_data(flag='test')
        if test:
            print('loading model')
            self.model.load_state_dict(torch.load(os.path.join('./checkpoints/'
+ setting, 'checkpoint.pth')))

        preds = []
        trues = []
        folder_path = './test_results/' + setting + '/'
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)

        self.model.eval()
        with torch.no_grad():
            for i, (batch_x, batch_y, batch_x_mark, batch_y_mark) in
enumerate(test_loader):
                batch_x = batch_x.float().to(self.device)
                batch_y = batch_y.float().to(self.device)

                batch_x_mark = batch_x_mark.float().to(self.device)
                batch_y_mark = batch_y_mark.float().to(self.device)

                # decoder input
                dec_inp = torch.zeros_like(batch_y[:, -self.args.pred_len:,
:]).float()
                dec_inp = torch.cat([batch_y[:, :self.args.label_len, :],
dec_inp], dim=1).float().to(self.device)
                # encoder - decoder
                if self.args.use_amp:
                    with torch.cuda.amp.autocast():
                        if self.args.output_attention:
                            outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)[0]
                        else:
                            outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)
                else:
                    if self.args.output_attention:
                        outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)[0]

                    else:
                        outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)

                f_dim = -1 if self.args.features == 'MS' else 0
                outputs = outputs[:, -self.args.pred_len:, f_dim:]
                batch_y = batch_y[:, -self.args.pred_len:,
f_dim:].to(self.device)
                outputs = outputs.detach().cpu().numpy()
                batch_y = batch_y.detach().cpu().numpy()

                pred = outputs  # outputs.detach().cpu().numpy()  # .squeeze()
                true = batch_y  # batch_y.detach().cpu().numpy()  # .squeeze()
                preds.append(pred)
                trues.append(true)
                if i % 20 == 0:
                    input = batch_x.detach().cpu().numpy()
                    gt = np.concatenate((input[0, :, -1], true[0, :, -1]),
axis=0)
                    pd = np.concatenate((input[0, :, -1], pred[0, :, -1]),
axis=0)
                    visual(gt, pd, os.path.join(folder_path, str(i) + '.pdf'))

        preds = np.array(preds)
        trues = np.array(trues)
        print('test shape:', preds.shape, trues.shape)
        preds = preds.reshape(-1, preds.shape[-2], preds.shape[-1])
        trues = trues.reshape(-1, trues.shape[-2], trues.shape[-1])
        print('test shape:', preds.shape, trues.shape)

        # result save
        folder_path = './results/' + setting + '/'
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)

        mae, mse, rmse, mape, mspe = metric(preds, trues)
        print('mse:{}, mae:{}'.format(mse, mae))
        f = open("result.txt", 'a')
        f.write(setting + "  \n")
        f.write('mse:{}, mae:{}'.format(mse, mae))
        f.write('\n')
        f.write('\n')
        f.close()

        np.save(folder_path + 'metrics.npy', np.array([mae, mse, rmse, mape,
mspe]))
        np.save(folder_path + 'pred.npy', preds)
        np.save(folder_path + 'true.npy', trues)

        return

    def predict(self, setting, load=False):
        pred_data, pred_loader = self._get_data(flag='pred')

        if load:
            path = os.path.join(self.args.checkpoints, setting)
            best_model_path = path + '/' + 'checkpoint.pth'
            self.model.load_state_dict(torch.load(best_model_path))

        preds = []

        self.model.eval()
        with torch.no_grad():
            for i, (batch_x, batch_y, batch_x_mark, batch_y_mark) in
enumerate(pred_loader):
                batch_x = batch_x.float().to(self.device)
                batch_y = batch_y.float()
                batch_x_mark = batch_x_mark.float().to(self.device)
                batch_y_mark = batch_y_mark.float().to(self.device)
                # decoder input
                dec_inp = torch.zeros([batch_y.shape[0], self.args.pred_len,
batch_y.shape[2]]).float()
                dec_inp = torch.cat([batch_y[:, :self.args.label_len, :],
dec_inp], dim=1).float().to(self.device)
                # encoder - decoder
                if self.args.use_amp:
                    with torch.cuda.amp.autocast():
                        if self.args.output_attention:
                            outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)[0]
                        else:
                            outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)
                else:
                    if self.args.output_attention:
                        outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)[0]
                    else:
                        outputs = self.model(batch_x, batch_x_mark, dec_inp,
batch_y_mark)
                pred = outputs.detach().cpu().numpy()  # .squeeze()
                preds.append(pred)

        preds = np.array(preds)
        preds = preds.reshape(-1, preds.shape[-2], preds.shape[-1])

        # result save
        folder_path = './results/' + setting + '/'
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)

        np.save(folder_path + 'real_prediction.npy', preds)

        return
```

避免无意义的warming

```
import warnings
warnings.filterwarnings('ignore')
```

性能指标代码(无需修改)

```
import numpy as np


def RSE(pred, true):
    return np.sqrt(np.sum((true - pred) ** 2)) / np.sqrt(np.sum((true -
true.mean()) ** 2))


def CORR(pred, true):
    u = ((true - true.mean(0)) * (pred - pred.mean(0))).sum(0)
    d = np.sqrt(((true - true.mean(0)) ** 2 * (pred - pred.mean(0)) **
2).sum(0))
    return (u / d).mean(-1)


def MAE(pred, true):
    return np.mean(np.abs(pred - true))


def MSE(pred, true):
    return np.mean((pred - true) ** 2)


def RMSE(pred, true):
    return np.sqrt(MSE(pred, true))


def MAPE(pred, true):
    return np.mean(np.abs((pred - true) / true))


def MSPE(pred, true):
    return np.mean(np.square((pred - true) / true))


def metric(pred, true):
    mae = MAE(pred, true)
    mse = MSE(pred, true)
    rmse = RMSE(pred, true)
    mape = MAPE(pred, true)
    mspe = MSPE(pred, true)

    return mae, mse, rmse, mape, mspe
```

画图的部分

常用的图例

1， 每种情况下性能指标的数值（一个数值）

2，预测曲线和实际的曲线

3，特定考量的属性，比如平稳性

4，算法的网络框图

数据的处理

```
from data_provider.data_loader import Dataset_ETT_hour, Dataset_ETT_minute,
Dataset_Custom, Dataset_Pred
from torch.utils.data import DataLoader

# 定义数据集
```

```python
data_dict = {
    'ETTh1': Dataset_ETT_hour,
    'ETTh2': Dataset_ETT_hour,
    'ETTm1': Dataset_ETT_minute,
    'ETTm2': Dataset_ETT_minute,
    'custom': Dataset_Custom,
}



def data_provider(args, flag):
    Data = data_dict[args.data]
    timeenc = 0 if args.embed != 'timeF' else 1

    if flag == 'test':# test
        shuffle_flag = False
        drop_last = True
        batch_size = args.batch_size
        freq = args.freq
    elif flag == 'pred': # pred
        shuffle_flag = False
        drop_last = False
        batch_size = 1
        freq = args.freq
        Data = Dataset_Pred
    else:
        shuffle_flag = True
        drop_last = True
        batch_size = args.batch_size
        freq = args.freq

    data_set = Data( # 导入数据集的信息
        root_path=args.root_path,
        data_path=args.data_path,
        flag=flag,
        size=[args.seq_len, args.label_len, args.pred_len],
        features=args.features,
        target=args.target,
        timeenc=timeenc,
        freq=freq
    )
    print(flag, len(data_set))
    data_loader = DataLoader( # 导入数据集的的一些参数
        data_set,
        batch_size=batch_size,
        shuffle=shuffle_flag,
        num_workers=args.num_workers,
        drop_last=drop_last)
    return data_set, data_loader
```

数据的读取

```python
import os
import numpy as np
import pandas as pd
import os
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import StandardScaler
from utils.timefeatures import time_features
import warnings

warnings.filterwarnings('ignore')

# 导入 EIT_hour的数据集
class Dataset_ETT_hour(Dataset):
    # 导入数据集
    def __init__(self, root_path, flag='train', size=None,
                 features='S', data_path='ETTh1.csv',
                 target='OT', scale=True, timeenc=0, freq='h'):
        # size [seq_len, label_len, pred_len]
        # info
        if size == None:
            self.seq_len = 24 * 4 * 4
            self.label_len = 24 * 4
            self.pred_len = 24 * 4
        else:
            self.seq_len = size[0]
            self.label_len = size[1]
            self.pred_len = size[2]
        # init
        assert flag in ['train', 'test', 'val']
        type_map = {'train': 0, 'val': 1, 'test': 2}
        self.set_type = type_map[flag]

        self.features = features
        self.target = target
        self.scale = scale
        self.timeenc = timeenc
        self.freq = freq

        self.root_path = root_path
        self.data_path = data_path
        self.__read_data__()

    def __read_data__(self):
        self.scaler = StandardScaler()
        # 导入数据
        df_raw = pd.read_csv(os.path.join(self.root_path,
                                          self.data_path))

        border1s = [0, 12 * 30 * 24 - self.seq_len, 12 * 30 * 24 + 4 * 30 * 24 -
self.seq_len]
        border2s = [12 * 30 * 24, 12 * 30 * 24 + 4 * 30 * 24, 12 * 30 * 24 + 8 *
30 * 24]
        border1 = border1s[self.set_type]
        border2 = border2s[self.set_type]

        # 导入所有的列，或者不份额的列
        if self.features == 'M' or self.features == 'MS':
            cols_data = df_raw.columns[1:]
            df_data = df_raw[cols_data]
        elif self.features == 'S':
            df_data = df_raw[[self.target]]
        if self.scale:
            train_data = df_data[border1s[0]:border2s[0]]
            self.scaler.fit(train_data.values)
            data = self.scaler.transform(df_data.values)
        else:
            data = df_data.values

        df_stamp = df_raw[['date']][border1:border2]
        df_stamp['date'] = pd.to_datetime(df_stamp.date)
        if self.timeenc == 0:
            df_stamp['month'] = df_stamp.date.apply(lambda row: row.month, 1)
            df_stamp['day'] = df_stamp.date.apply(lambda row: row.day, 1)
            df_stamp['weekday'] = df_stamp.date.apply(lambda row: row.weekday(),
1)
            df_stamp['hour'] = df_stamp.date.apply(lambda row: row.hour, 1)
            data_stamp = df_stamp.drop(['date'], 1).values
        elif self.timeenc == 1:
            data_stamp = time_features(pd.to_datetime(df_stamp['date'].values),
freq=self.freq)
            data_stamp = data_stamp.transpose(1, 0)

        self.data_x = data[border1:border2]
        self.data_y = data[border1:border2]
        self.data_stamp = data_stamp

    def __getitem__(self, index):
        s_begin = index
        s_end = s_begin + self.seq_len
        r_begin = s_end - self.label_len
        r_end = r_begin + self.label_len + self.pred_len

        seq_x = self.data_x[s_begin:s_end]
        seq_y = self.data_y[r_begin:r_end]
        seq_x_mark = self.data_stamp[s_begin:s_end]
        seq_y_mark = self.data_stamp[r_begin:r_end]

        return seq_x, seq_y, seq_x_mark, seq_y_mark

    def __len__(self):
        return len(self.data_x) - self.seq_len - self.pred_len + 1

    def inverse_transform(self, data):
        return self.scaler.inverse_transform(data)


class Dataset_ETT_minute(Dataset):
    def __init__(self, root_path, flag='train', size=None,
                 features='S', data_path='ETTm1.csv',
                 target='OT', scale=True, timeenc=0, freq='t'):
        # size [seq_len, label_len, pred_len]
        # info
        if size == None:
            self.seq_len = 24 * 4 * 4
            self.label_len = 24 * 4
            self.pred_len = 24 * 4
        else:
            self.seq_len = size[0]
            self.label_len = size[1]
            self.pred_len = size[2]
        # init
        assert flag in ['train', 'test', 'val']
        type_map = {'train': 0, 'val': 1, 'test': 2}
        self.set_type = type_map[flag]

        self.features = features
        self.target = target
        self.scale = scale
        self.timeenc = timeenc
        self.freq = freq

        self.root_path = root_path
        self.data_path = data_path
        self.__read_data__()

    def __read_data__(self):
        self.scaler = StandardScaler()
        df_raw = pd.read_csv(os.path.join(self.root_path,
                                          self.data_path))

        border1s = [0, 12 * 30 * 24 * 4 - self.seq_len, 12 * 30 * 24 * 4 + 4 *
30 * 24 * 4 - self.seq_len]
        border2s = [12 * 30 * 24 * 4, 12 * 30 * 24 * 4 + 4 * 30 * 24 * 4, 12 *
30 * 24 * 4 + 8 * 30 * 24 * 4]
        border1 = border1s[self.set_type]
        border2 = border2s[self.set_type]

        if self.features == 'M' or self.features == 'MS':
            cols_data = df_raw.columns[1:]
            df_data = df_raw[cols_data]
        elif self.features == 'S':
            df_data = df_raw[[self.target]]

        if self.scale:
            train_data = df_data[border1s[0]:border2s[0]]
            self.scaler.fit(train_data.values)
            data = self.scaler.transform(df_data.values)
        else:
            data = df_data.values

        df_stamp = df_raw[['date']][border1:border2]
        df_stamp['date'] = pd.to_datetime(df_stamp.date)
        if self.timeenc == 0:
            df_stamp['month'] = df_stamp.date.apply(lambda row: row.month, 1)
            df_stamp['day'] = df_stamp.date.apply(lambda row: row.day, 1)
            df_stamp['weekday'] = df_stamp.date.apply(lambda row: row.weekday(),
1)
            df_stamp['hour'] = df_stamp.date.apply(lambda row: row.hour, 1)
            df_stamp['minute'] = df_stamp.date.apply(lambda row: row.minute, 1)
            df_stamp['minute'] = df_stamp.minute.map(lambda x: x // 15)
            data_stamp = df_stamp.drop(['date'], 1).values
        elif self.timeenc == 1:
            data_stamp = time_features(pd.to_datetime(df_stamp['date'].values),
freq=self.freq)
            data_stamp = data_stamp.transpose(1, 0)
```

```python
        self.data_x = data[border1:border2]
        self.data_y = data[border1:border2]
        self.data_stamp = data_stamp

    def __getitem__(self, index):
        s_begin = index
        s_end = s_begin + self.seq_len
        r_begin = s_end - self.label_len
        r_end = r_begin + self.label_len + self.pred_len

        seq_x = self.data_x[s_begin:s_end]
        seq_y = self.data_y[r_begin:r_end]
        seq_x_mark = self.data_stamp[s_begin:s_end]
        seq_y_mark = self.data_stamp[r_begin:r_end]

        return seq_x, seq_y, seq_x_mark, seq_y_mark

    def __len__(self):
        return len(self.data_x) - self.seq_len - self.pred_len + 1

    def inverse_transform(self, data):
        return self.scaler.inverse_transform(data)


class Dataset_Custom(Dataset):
    def __init__(self, root_path, flag='train', size=None,
                 features='S', data_path='ETTh1.csv',
                 target='OT', scale=True, timeenc=0, freq='h'):
        # size [seq_len, label_len, pred_len]
        # info
        if size == None:
            self.seq_len = 24 * 4 * 4
            self.label_len = 24 * 4
            self.pred_len = 24 * 4
        else:
            self.seq_len = size[0]
            self.label_len = size[1]
            self.pred_len = size[2]
        # init
        assert flag in ['train', 'test', 'val']
        type_map = {'train': 0, 'val': 1, 'test': 2}
        self.set_type = type_map[flag]

        self.features = features
        self.target = target
        self.scale = scale
        self.timeenc = timeenc
        self.freq = freq

        self.root_path = root_path
        self.data_path = data_path
        self.__read_data__()

    def __read_data__(self):
        self.scaler = StandardScaler()
        df_raw = pd.read_csv(os.path.join(self.root_path,
                                          self.data_path))
        '''
        df_raw.columns: ['date', ...(other features), target feature]
        '''
        cols = list(df_raw.columns)
        cols.remove(self.target)
        cols.remove('date')
        df_raw = df_raw[['date'] + cols + [self.target]]
        # print(cols)
        num_train = int(len(df_raw) * 0.7)
        num_test = int(len(df_raw) * 0.2)
        num_vali = len(df_raw) - num_train - num_test
        border1s = [0, num_train - self.seq_len, len(df_raw) - num_test -
self.seq_len]
        border2s = [num_train, num_train + num_vali, len(df_raw)]
        border1 = border1s[self.set_type]
        border2 = border2s[self.set_type]

        if self.features == 'M' or self.features == 'MS':
            cols_data = df_raw.columns[1:]
            df_data = df_raw[cols_data]
        elif self.features == 'S':
            df_data = df_raw[[self.target]]

        if self.scale:
            train_data = df_data[border1s[0]:border2s[0]]
            self.scaler.fit(train_data.values)
            data = self.scaler.transform(df_data.values)
        else:
            data = df_data.values

        df_stamp = df_raw[['date']][border1:border2]
        df_stamp['date'] = pd.to_datetime(df_stamp.date)
        if self.timeenc == 0:
            df_stamp['month'] = df_stamp.date.apply(lambda row: row.month, 1)
            df_stamp['day'] = df_stamp.date.apply(lambda row: row.day, 1)
            df_stamp['weekday'] = df_stamp.date.apply(lambda row: row.weekday(),
1)
            df_stamp['hour'] = df_stamp.date.apply(lambda row: row.hour, 1)
            data_stamp = df_stamp.drop(['date'], 1).values
        elif self.timeenc == 1:
            data_stamp = time_features(pd.to_datetime(df_stamp['date'].values),
freq=self.freq)
            data_stamp = data_stamp.transpose(1, 0)

        self.data_x = data[border1:border2]
        self.data_y = data[border1:border2]
        self.data_stamp = data_stamp

    def __getitem__(self, index):
        s_begin = index
        s_end = s_begin + self.seq_len
        r_begin = s_end - self.label_len
        r_end = r_begin + self.label_len + self.pred_len

        seq_x = self.data_x[s_begin:s_end]
        seq_y = self.data_y[r_begin:r_end]
        seq_x_mark = self.data_stamp[s_begin:s_end]
        seq_y_mark = self.data_stamp[r_begin:r_end]
        return seq_x, seq_y, seq_x_mark, seq_y_mark

    def __len__(self):
        return len(self.data_x) - self.seq_len - self.pred_len + 1

    def inverse_transform(self, data):
        return self.scaler.inverse_transform(data)


class Dataset_Pred(Dataset):
    def __init__(self, root_path, flag='pred', size=None,
                 features='S', data_path='ETTh1.csv',
                 target='OT', scale=True, inverse=False, timeenc=0,
freq='15min', cols=None):
        # size [seq_len, label_len, pred_len]
        # info
        if size == None:
            self.seq_len = 24 * 4 * 4
            self.label_len = 24 * 4
            self.pred_len = 24 * 4
        else:
            self.seq_len = size[0]
            self.label_len = size[1]
            self.pred_len = size[2]
        # init
        assert flag in ['pred']

        self.features = features
        self.target = target
        self.scale = scale
        self.inverse = inverse
        self.timeenc = timeenc
        self.freq = freq
        self.cols = cols
        self.root_path = root_path
        self.data_path = data_path
        self.__read_data__()

    def __read_data__(self):
        self.scaler = StandardScaler()
        df_raw = pd.read_csv(os.path.join(self.root_path,
                                          self.data_path))
        '''
        df_raw.columns: ['date', ...(other features), target feature]
        '''
        if self.cols:
            cols = self.cols.copy()
            cols.remove(self.target)
        else:
            cols = list(df_raw.columns)
            cols.remove(self.target)
            cols.remove('date')
        df_raw = df_raw[['date'] + cols + [self.target]]
        border1 = len(df_raw) - self.seq_len
        border2 = len(df_raw)

        if self.features == 'M' or self.features == 'MS':
            cols_data = df_raw.columns[1:]
            df_data = df_raw[cols_data]
        elif self.features == 'S':
            df_data = df_raw[[self.target]]

        if self.scale:
            self.scaler.fit(df_data.values)
            data = self.scaler.transform(df_data.values)
        else:
            data = df_data.values

        tmp_stamp = df_raw[['date']][border1:border2]
        tmp_stamp['date'] = pd.to_datetime(tmp_stamp.date)
        pred_dates = pd.date_range(tmp_stamp.date.values[-1],
periods=self.pred_len + 1, freq=self.freq)

        df_stamp = pd.DataFrame(columns=['date'])
        df_stamp.date = list(tmp_stamp.date.values) + list(pred_dates[1:])
        if self.timeenc == 0:
            df_stamp['month'] = df_stamp.date.apply(lambda row: row.month, 1)
            df_stamp['day'] = df_stamp.date.apply(lambda row: row.day, 1)
            df_stamp['weekday'] = df_stamp.date.apply(lambda row: row.weekday(),
1)
            df_stamp['hour'] = df_stamp.date.apply(lambda row: row.hour, 1)
            df_stamp['minute'] = df_stamp.date.apply(lambda row: row.minute, 1)
            df_stamp['minute'] = df_stamp.minute.map(lambda x: x // 15)
            data_stamp = df_stamp.drop(['date'], 1).values
        elif self.timeenc == 1:
            data_stamp = time_features(pd.to_datetime(df_stamp['date'].values),
freq=self.freq)
            data_stamp = data_stamp.transpose(1, 0)

        self.data_x = data[border1:border2]
        if self.inverse:
            self.data_y = df_data.values[border1:border2]
        else:
            self.data_y = data[border1:border2]
        self.data_stamp = data_stamp

    def __getitem__(self, index):
        s_begin = index
        s_end = s_begin + self.seq_len
        r_begin = s_end - self.label_len
        r_end = r_begin + self.label_len + self.pred_len

        seq_x = self.data_x[s_begin:s_end]
        if self.inverse:
            seq_y = self.data_x[r_begin:r_begin + self.label_len]
        else:
            seq_y = self.data_y[r_begin:r_begin + self.label_len]
        seq_x_mark = self.data_stamp[s_begin:s_end]
        seq_y_mark = self.data_stamp[r_begin:r_end]

        return seq_x, seq_y, seq_x_mark, seq_y_mark

    def __len__(self):
        return len(self.data_x) - self.seq_len + 1
    def inverse_transform(self, data):
        return self.scaler.inverse_transform(data)
```

使用pbar来显示运行进度

参考：https://github.com/tqdm/tqdm

瓜分数据集

```
def data_div(data, train: float=0.8, validation: float=0.1, test: float=0.1):
    numTrain = math.floor(data.size*train)
    numVal = math.floor(data.size*validation)
    dataTrain = data[:numTrain]
    dataVal = data[numTrain:numTrain+numVal]
    dataTest = data[numTrain+numVal:]
    return dataTrain, dataVal, dataTest
```

对训练集归一化处理

```
def normalization(x):
    sc = MinMaxScaler(feature_range=(0,1))
    sc.fit(x)
    x_norm = sc.transform(x)
    return x_norm, sc
```

如何画出放大序列局部的图

如何利用时序图来描绘时间序列数据集的分布

时间序列数据集的描述和处理

pandas的基本使用说明

参考教程:pandas一个简单的介绍

0, pandas的一个基础

查看pandas的版本

```
import pandas as pd
pandas.__version__
>'0.24.0'
```

查看pandas Namesapce里面的内容

```
pd.<TAB>
```

查看Pandas的内建文件

```
pd?
```

建立pandas的 Series对象

```
data=pd.Series([0.25, 0.5, 0.75, 1.0])
data
>data=
0   0.25
1   0.5
2   0.75
3   1.0
dtype: float64
# o,1,2,3:index
# 0.25, 0.5, 0.75, 1:value
```

使用**values, index**

```
data.values
>array([0.25, 0.5, 0.75,1])
data.index
>RangeIndex(start=0, stop=4, step=1)
```

访问单个元素

```
data[1]
>0.5
data[1:3]
> 1 0.5
  2 0.75
dtype: float64
```

设置不同的index,比如字母

```
data=pd.Series([0.25,0.5, 0.75,1.0],
               index=['a','b','c','d'])
data
>
a 0.25
b 0.5
c 0.75
d 1.00
dtype: float64
```

通过index来访问元素

```
data['b']
>
0.5
```

使用不连续的index

```
data=pd.Series([0.25,0.5,0.75,1.0],
              index=[2,5,3,7])
data
>
2 0.25
5 0.5
3 0.75
7 1.00
dtype: float64
>data[5]
0.5
```

使用Series来创建特殊的字典

```
population_dict={'California':38,
                'Texas':20,
                'New York',19,
                'Florian',12,
                'Tillinios',14}
population=pd.Series(population_dict)
population
>
California 38
Texas 20
New York 19
Florian 12
Tillinios 14
```

访问对应的文档

```
population['California']
>
California
-------------------------------------
population['California':'Tillinios']
>
California 38
Texas 20
New York 19
Florian 12
Tillinios 14
```

创建对应的字典

```
pd.Series({2:'a',1:'b',3:'c'})
2 a
1 b
3 c
```

访问多个index

```
pd.Series({2:'a',1:'b',3:'c'},index=[3,2])
>
3 c
2 a
dtype: object
```

创建一个多列的Dataframe

```
population_dict={'California':1,
                'Texas':23,
                'New York': 19,
                'Floria':10,
                'Illinois':12}
population=pd.Series(population_dict)
area_dict={'California':43,
          'Texas':41,
          'New York':32,
          'Floria':11,
          'Illinois': 21}
area=pd.Series(area_dict)
states=pd.DataFrame(['population':population,
                    'area',area])
state
```

	population	area
California	1	43
Texas	23	41
New York	19	32
Floria	10	11
Illinois	12	21

```
states.column
>
Index(['population','area'],dtype='object')
-------------------------------------------
states['area']
California 43
Texas 41
New York 32
Floria 11
Illinois 21
```

创建这样的DataFrame

```
pd.DataFrame(np.random.rand(3,2),
            columns=['foo','bar'],
            index=['a','b','c'])
```

	foo	bar
a	0.8	0.21
b	0.44	0.1
c	0.04	0.905

创建Pandas的index

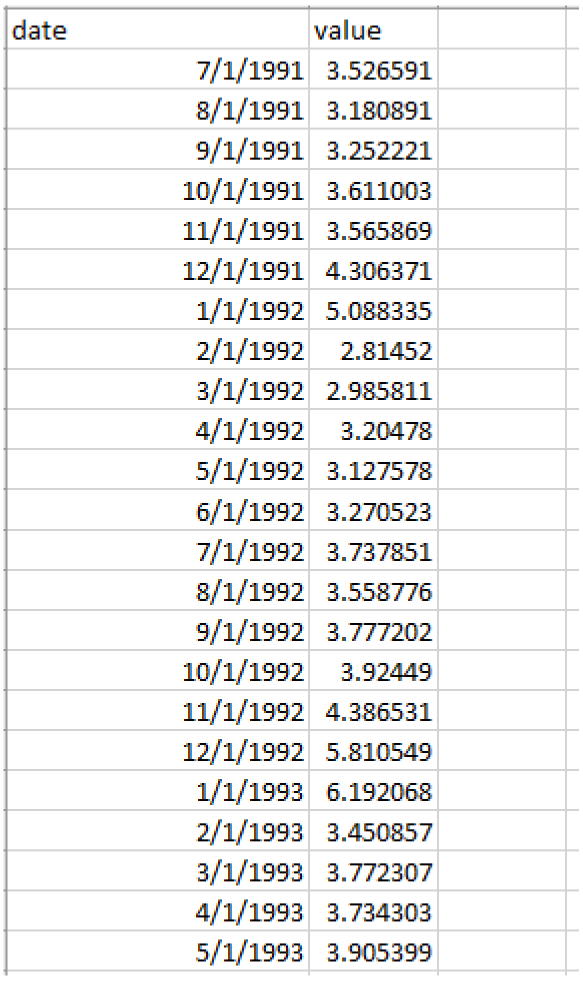
```
ind=pd.Index([2,3,5,7,11])
ind
>
Int6Index([2,3,5,7,11],dtype='int64')
```

获取index的一些属性

```
print(ind.size,ind.shape,ind.ndim,ind.dtype)
>
5, (5,) 1 int64
```

1，读取时间序列数据

数据类型：

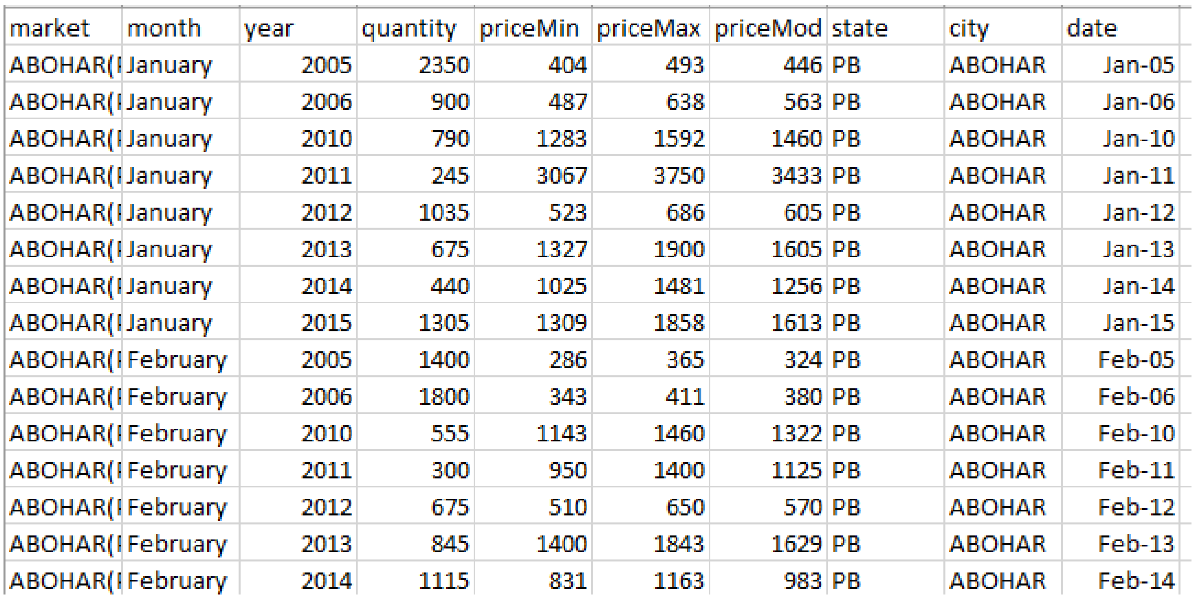


date	value
7/1/1991	3.526591
8/1/1991	3.180891
9/1/1991	3.252221
10/1/1991	3.611003
11/1/1991	3.565869
12/1/1991	4.306371
1/1/1992	5.088335
2/1/1992	2.81452
3/1/1992	2.985811
4/1/1992	3.20478
5/1/1992	3.127578
6/1/1992	3.270523
7/1/1992	3.737851
8/1/1992	3.558776
9/1/1992	3.777202
10/1/1992	3.92449
11/1/1992	4.386531
12/1/1992	5.810549
1/1/1993	6.192068
2/1/1993	3.450857
3/1/1993	3.772307
4/1/1993	3.734303
5/1/1993	3.905399

```
from dateutil.parser import parse
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
plt.rcParams.update({'figure.figsize': (10, 7), 'figure.dpi': 120})

# Import as Dataframe
df =
pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/a10.csv',
parse_dates=['date'])
# parse_dates=['date']将会是date这个column变成一个date field
df.head(10)
# 读取前10行数据

ser =
pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/a10.csv',
parse_dates=['date'], index_col='date')
# index_col会让date这一列变成index，因为csv文件中，没有index这个概念
ser.head()
```

2, panel data是基于时间的数据集

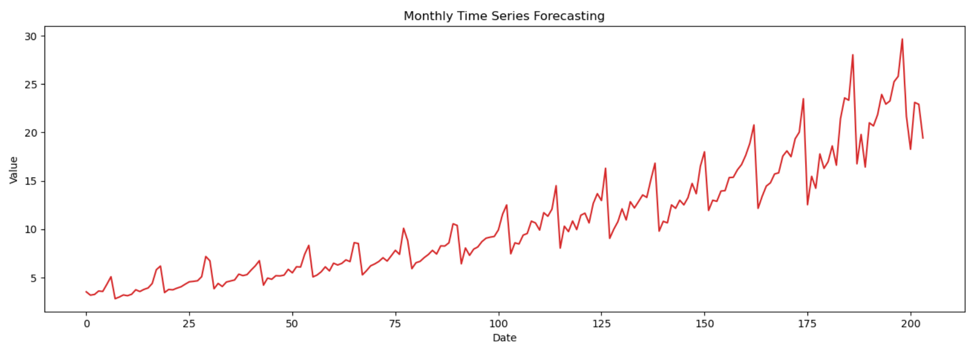
panel data相比于时间序列，对于同一个时期，会有一个或者多个变量



market	month	year	quantity	priceMin	priceMax	priceMod	state	city	date
ABOHAR(	January	2005	2350	404	493	446	PB	ABOHAR	Jan-05
ABOHAR(	January	2006	900	487	638	563	PB	ABOHAR	Jan-06
ABOHAR(	January	2010	790	1283	1592	1460	PB	ABOHAR	Jan-10
ABOHAR(	January	2011	245	3067	3750	3433	PB	ABOHAR	Jan-11
ABOHAR(	January	2012	1035	523	686	605	PB	ABOHAR	Jan-12
ABOHAR(	January	2013	675	1327	1900	1605	PB	ABOHAR	Jan-13
ABOHAR(	January	2014	440	1025	1481	1256	PB	ABOHAR	Jan-14
ABOHAR(	January	2015	1305	1309	1858	1613	PB	ABOHAR	Jan-15
ABOHAR(	February	2005	1400	286	365	324	PB	ABOHAR	Feb-05
ABOHAR(	February	2006	1800	343	411	380	PB	ABOHAR	Feb-06
ABOHAR(	February	2010	555	1143	1460	1322	PB	ABOHAR	Feb-10
ABOHAR(	February	2011	300	950	1400	1125	PB	ABOHAR	Feb-11
ABOHAR(	February	2012	675	510	650	570	PB	ABOHAR	Feb-12
ABOHAR(	February	2013	845	1400	1843	1629	PB	ABOHAR	Feb-13
ABOHAR(	February	2014	1115	831	1163	983	PB	ABOHAR	Feb-14

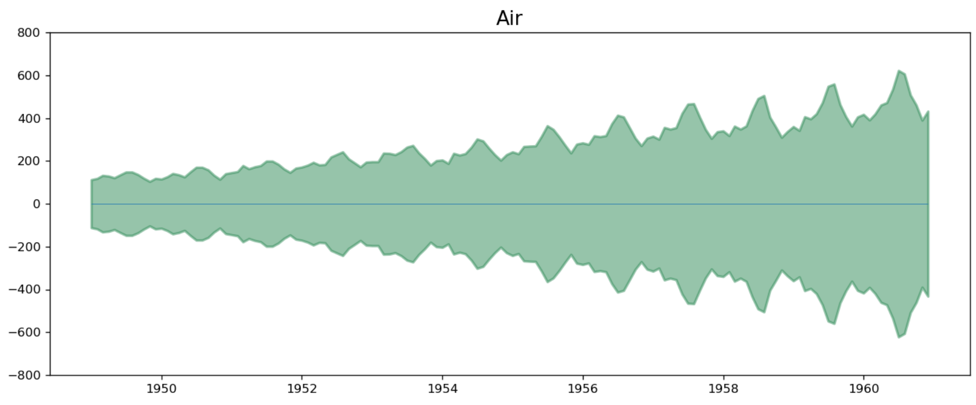
3, 可视化时间序列

```python
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd

df=pd.read_csv('a10.csv',parse_dates=['date'])

def plot_df(df,x,y,title="",xlabel='Date',ylabel='Value',dpi=100):
    fig=plt.figure(figsize=(16,5),dpi=dpi)
    plt.plot(x,y,color='tab:red')
    plt.gca().set(title=title,xlabel=xlabel,ylabel=ylabel)
    plt.show()
    fig.savefig('Ex1.png')

fig1=plot_df(df,x=df.index,y=df.value,title='Monthly Time Series Forecasting')
```



4, Two-side 图用于分析不同的年份的数据

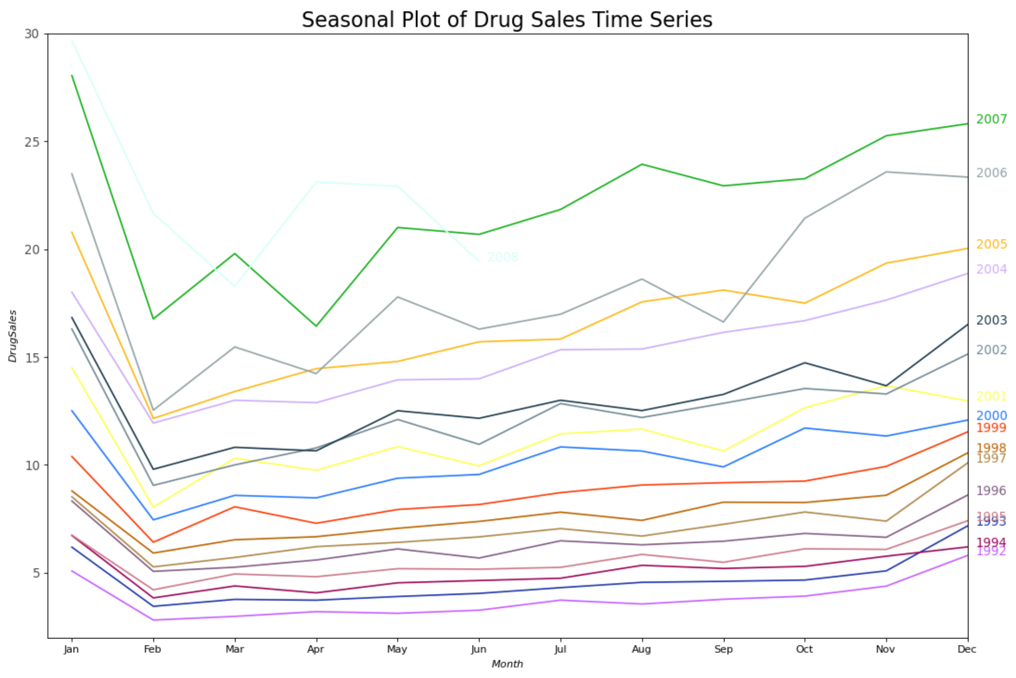
```python
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 读取数据
df = pd.read_csv('AirPassengers.csv',parse_dates=['date'])
# 获取时间数据
x=df['date'].values
# 获取对应的数值数据
y1=df['value'].values

fig, ax=plt.subplots(1,1,figsize=(16,5),dpi=120)
# 画two-side 图
plt.fill_between(x,y1=y1,y2=-y1,alpha=0.5,linewidth=2,color='seagreen')
plt.ylim(-800,800)
plt.title('Air',fontsize=16)
plt.hlines(y=0,xmin=np.min(df.date),xmax=np.max(df.date),linewidth=.5)
plt.show()
fig.savefig('Ex2.png')
```



5, 时间序列根据不同的季度来画图

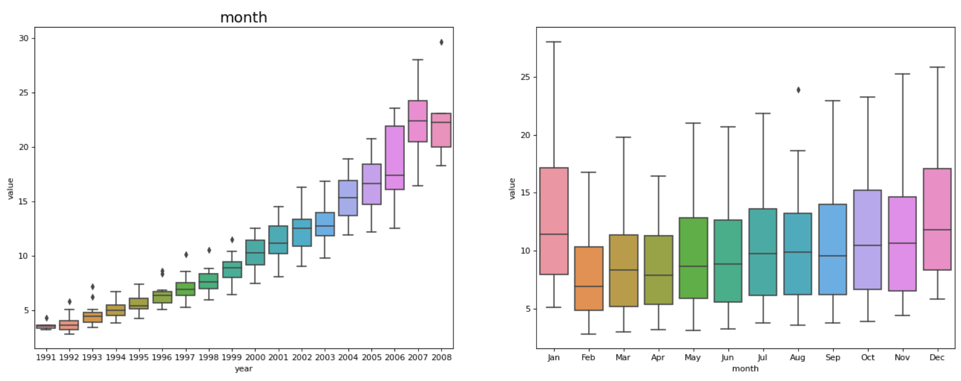
```python
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 读取数据
df=pd.read_csv('a10.csv',parse_dates=['date'])
# 重置index term
# inplace=True:不创建新的对象，直接对原对象进行修改
df.reset_index(inplace=True)
# 显示前10行
df.head(10)
```

```
df['year']=[d.year for d in df.date]
# strftime: datetime to string
df['month']=[d.strftime('%b') for d in df.date]
# unique: 返回不重复的年份
years=df['year'].unique()

np.random.seed(100)
# np.random.choice: 随机抽取数字，返回指定大小（size）的数组
# replac=False表示不可以抽取相同数字
mycolors=np.random.choice(list(mpl.colors.XKCD_COLORS.keys()),len(years),replace
=False)

fig=plt.figure(figsize=(16,12),dpi=80)

# enumerate: return count, value
for i, y in enumerate(years):
    if i > 0:
        plt.plot('month', 'value', data=df.loc[df.year==y, :],
color=mycolors[i], label=y)
        plt.text(df.loc[df.year==y, :].shape[0]-.9, df.loc[df.year==y, 'value']
[-1:].values[0], y, fontsize=12, color=mycolors[i])

# Decoration
plt.gca().set(xlim=(-0.3, 11), ylim=(2, 30), ylabel='$Drug Sales$',
xlabel='$Month$')
plt.yticks(fontsize=12, alpha=.7)
plt.title("Seasonal Plot of Drug Sales Time Series", fontsize=20)
plt.show()
fig.savefig('Ex3.png')
```



6,boxplot 体现年度和月度的分布

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns

df=pd.read_csv('a10.csv',parse_dates=['date'])
# 重设index,之前的index变为一列
# inplace=True: 不再生成新的Series
df.reset_index(inplace=True)

df['year']=[d.year for d in df.date]
# strftime: 获取data, 去掉重复的时间
df['month'] =[d.strftime('%b') for d in df.date]
years = df['year'].unique()

fig,axes=plt.subplots(1,2,figsize=(20,7),dpi=80)
sns.boxplot(x='year',y='value',data=df,ax=axes[0])
sns.boxplot(x='month',y='value',data=df.loc[~df.year.isin([1991,2008]),:])

axes[0].set_title('year',fontsize=18)
axes[0].set_title('month',fontsize=18)
plt.show()
fig.savefig('Ex5.png')
```



6.1 Advantage and draw back of this

Advantage: 可以很直观的查看年度或者每个月的分布，可以去辨识pattern的相似的地方

6.2 相关的解释

The boxplots make the year-wise and month-wise distributions evident. Also, in a month-wise boxplot, the months of December and January clearly has higher drug sales, which can be attributed to the holiday discounts season.

7, 如何查看时间序列的模式

Time Series = base level + Trend + Seasonality + Error

```

import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns

fig, axes = plt.subplots(1,3, figsize=(20,4), dpi=100)
pd.read_csv('guinearice.csv',
            parse_dates=['date'],
            index_col='date'). \
    plot(title='Trend Only', legend=False, ax=axes[0])
# legend= False: 去掉 legend
# ax=axes[0] 图1
pd.read_csv('sunspotarea.csv',
            parse_dates=['date'],
            index_col='date').\
    plot(title='Seasonality Only', legend=False, ax=axes[1])

pd.read_csv('AirPassengers.csv',
            parse_dates=['date'],
            index_col='date').\
    plot(title='Trend and Seasonality', legend=False, ax=axes[2])

plt.show()
fig.savefig('Ex6.png')
```

7.1 Advantage and drawbacks of this

A: 一个时间序列的pattern可以通过这种方式观察到

D: 并不是所有的时间序列都有trend 或者seasonality

7.2 相关的解释

A trend is observed when there is an increasing or decreasing slope observed in the time series. Whereas seasonality is observed when there is a distinct repeated pattern observed between regular intervals due to seasonal factors. It could be because of the month of the year, the day of the month, weekdays or even time of the day.

However, It is not mandatory that all time series must have a trend and/or seasonality. A time series may not have a distinct trend but have a seasonality. The opposite can also be true.

So, a time series may be imagined as a combination of the trend, seasonality and the error terms.

7.3 Attention: cyclic behaviour

Another aspect to consider is the **cyclic** behaviour. It happens when the rise and fall pattern in the series does not happen in fixed calendar-based intervals. Care should be taken to not confuse ‘cyclic’ effect with ‘seasonal’ effect.

So, How to diffentiate between a ‘cyclic’ vs ‘seasonal’ pattern?

If the patterns are not of fixed calendar based frequencies, then it is cyclic. Because, unlike the seasonality, cyclic effects are typically influenced by the business and other socio-economic factors.

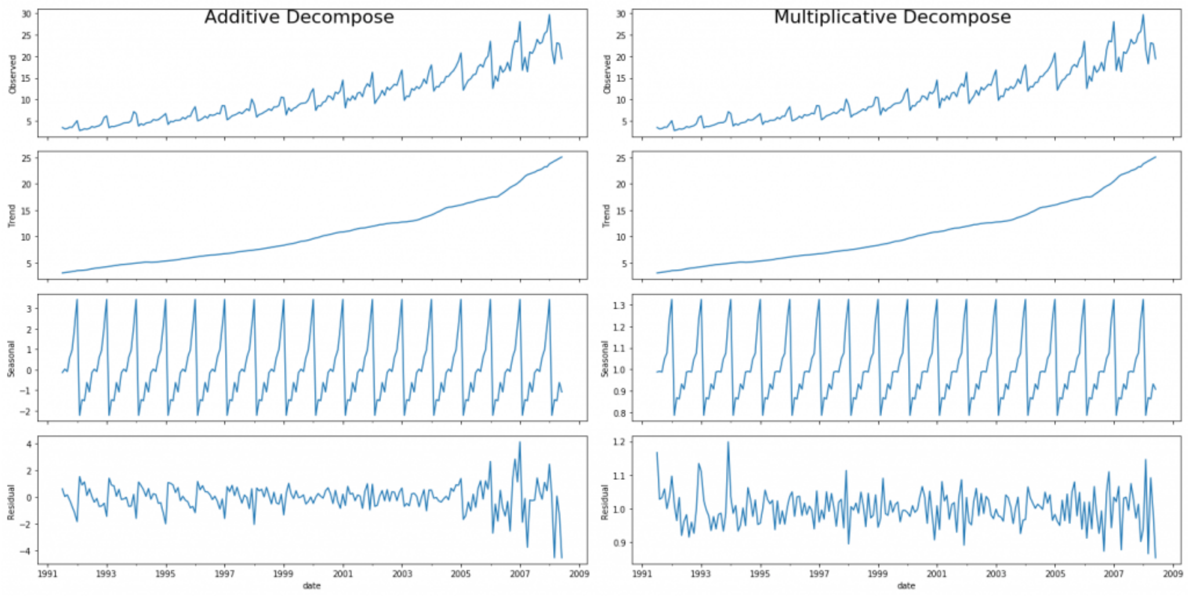
8, Additive and Multiplicative 时间序列

Additiive: Value = Base level + Trend+ Seasonality + Error

Multiplicative: Value = Base level X Trend XSeasonality XError

9，如何分解Time Series为多个分量

```
from statsmodels.tsa.seasonal import
seasonal_decompose
import pandas as pd
from dateutil.parser import parse
df=pd.read_csv('a10.csv',parse_dates=['date'], \
               index_col='date')

result_mul=seasonal_decompose(df['value'],
                              model='multiplicative',
                              extrapolate_trend='freq')
# extrapolate_trend='freq'
# 考虑在trend里面丢失的值或者在序列开始的误差
result_add=seasonal_decompose(df['value'],
                              model='additive',
                              extrapolate_trend='freq')
plt.rcParams.update({'figure.figsize':(10,10)})
result_mul.plot().suptitle('Multiplciative_Decompose',
                           fontsize=12)
result_add.plot().suptitle('Additive Decompose',
                           fontsize=12)
plt.show()
```



```
# Extract the Components ----
# Actual Values = Product of (Seasonal * Trend * Resid)
# 把seasonal, trend, resid, observed 四个分量放在一个dataFrame里面
df_reconstructed = pd.concat([result_mul.seasonal, result_mul.trend,
result_mul.resid, result_mul.observed], axis=1)
df_reconstructed.columns = ['seas', 'trend', 'resid', 'actual_values']
df_reconstructed.head()
```

9.1 Advantages and drawbacks of this

Advantage of multiplicatie decomposition: 仔细观察additive decomposition当中的residual，还是会有一些pattern, 但是multiplicative decomposition则非常随机，所以更好

9.2 相关的解释

If you look at the residuals of the additive decomposition closely, it has some pattern left over. The multiplicative decomposition, however, looks quite random which is good. So ideally, multiplicative decomposition should be preferred for this particular series.

10， stationary， non-stationary 时间序列

stationary series不依赖于时间，所以统计特性，比如mean, variance, autocorreation都是常数，其中Autocorrelation of the series is nothing but the correlation of the series with its previous values, more on this coming up. Stationary TS 没有seasonal effect,大部分的非平稳的时间序列都可以使用一些变换去变成平稳的时间序列。

11, 如何让一个nonstationary 的时间序列变成一个stationary的时间序列

1， 对时间序列使用查分，一次或者多次

2，对时间序列取log

3, 对时间序列取n次方根

4，上面的方法的组合

比如一个时间序列：

[1,5,2,12,20]

第一次差分

[4,-3,10,8]

第二次差分

[-7,-3,-2]

11.1 为什么要把一个nonstationary TS变成一个stationary TS

1, stationary的时间序列更加容易并且预测更加可靠

2，自回归模型通常是线性自回归模型，使用原本序列的延迟作为predictor. 线性回归模型工作如果变量是不相关的，所以stationariziing可以让序列的变量不相关，从而是的predictor(lag of the series)近乎独立

12 如何测试stationarity

1， 直接画出来

2，将时间序列分离为不同的分量，然后计算他们的统计量，比如mean, variance, the autocorrelation. 如果这些统计量，是不同的，则这个序列不太可能是stationary的。

3，数值方法，叫做“unit Root Test"

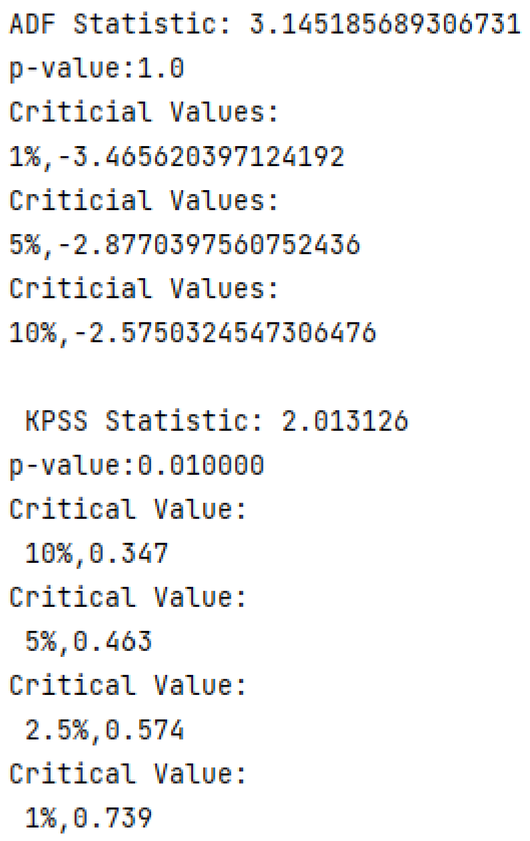
3.1 Augumented Dickey Fuller Test (ADH Test) **most common** the null hypothesis is the time series possesses a unit root and is non-stationary. So, id the P-Value in ADH test is less than the significance level (0.05), you reject the null hypothesis.

3.2 Kwiatkowshi-Philips-Schmidt-Shin-KPSS test (trend stationary)

The KPSS test, on the other hand, is used to test for trend stationarity. The null hypothesis and the P-Value interpretation is just the opposite of ADH test.

3.3 Philips Perron test (PP Test)

```
import pandas as pd
from statsmodels.tsa.stattools import adfuller, kpss
## ADF Test
df=pd.read_csv('a10.csv',parse_dates=['date'])
result=adfuller(df.value.values,autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value:{result[1]}')
for key,value in result[4].items():
    print('Criticial Values:')
    print(f'{key},{value}')

## KPSS Test
result=kpss(df.value.values,regression='c')
print('\n KPSS Statistic: %f' %result[0])
print('p-value:%f' % result[1])
for key,value in result[3].items():
    print('Critical Value:')
    print(f' {key},{value}')
```

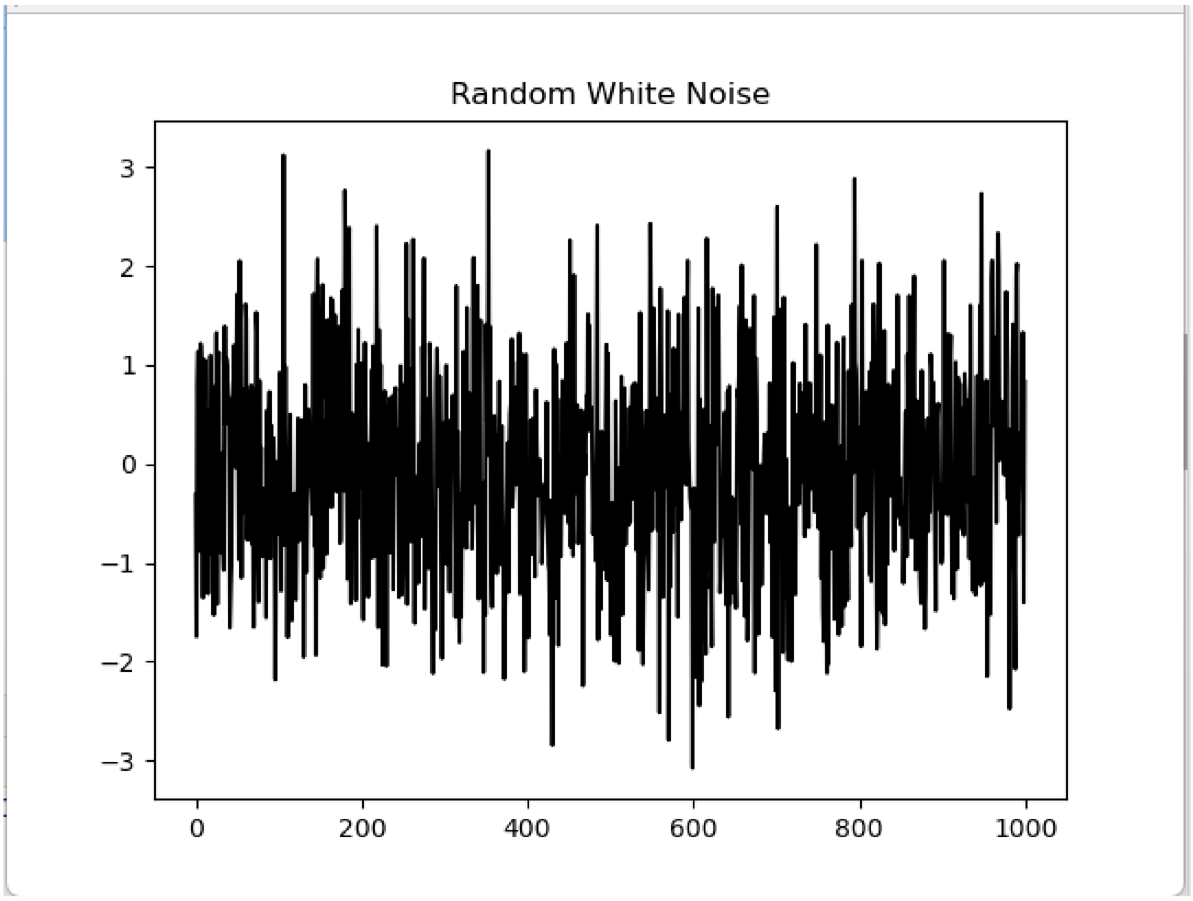


```
ADF Statistic: 3.145185689306731
p-value:1.0
Criticial Values:
1%,-3.465620397124192
Criticial Values:
5%,-2.8770397560752436
Criticial Values:
10%,-2.5750324547306476

 KPSS Statistic: 2.013126
p-value:0.010000
Critical Value:
 10%,0.347
Critical Value:
 5%,0.463
Critical Value:
 2.5%,0.574
Critical Value:
 1%,0.739
```

13, white noise 没有pattern,

white noise 也不是时间的函数，但是white noise 的mean 确定是0

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

randvals=np.random.randn(1000)
pd.Series(randvals).plot(title='Random White Noise',color='k')
plt.show()
```



14, 如何给时间序列去掉趋势分量

1， 时间序列减去 line of best fit.

The line of best fit may be obtained from a linear regression model with the time steps as the predictor. For more complex trends, you may want to use quadratic terms (x^2) in the model.

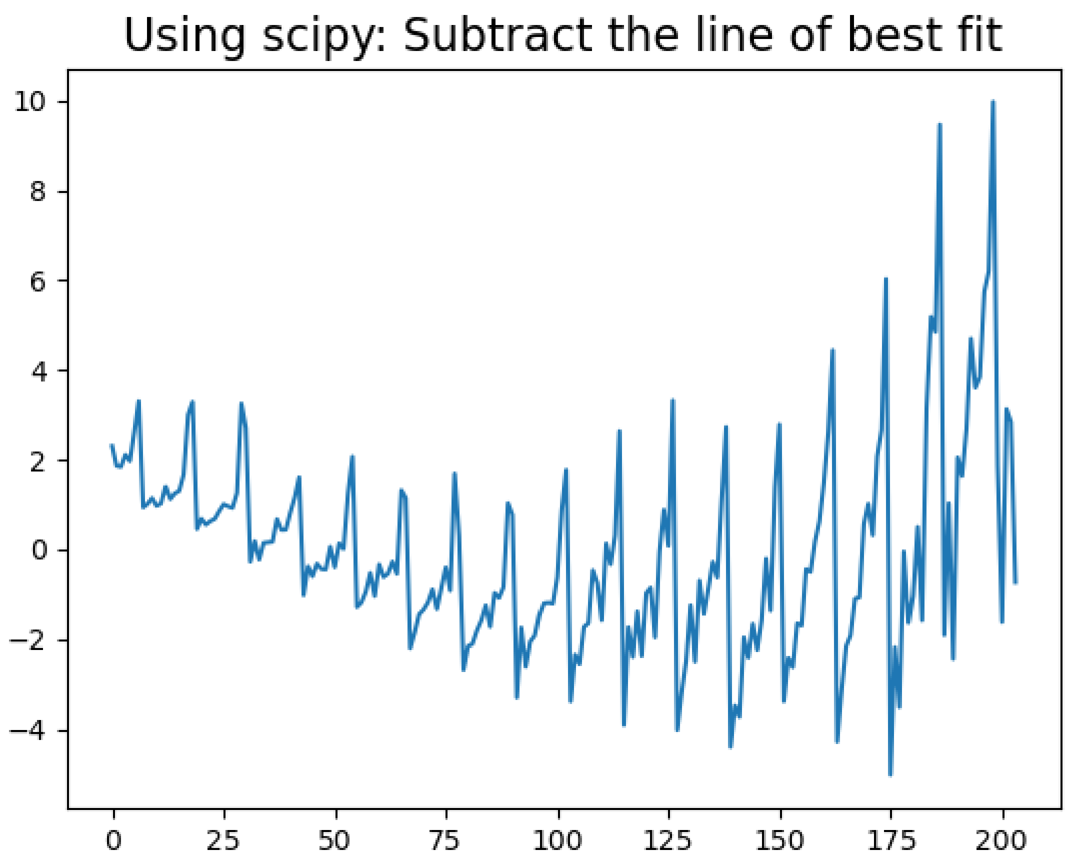
2，减去时间序列的trend component

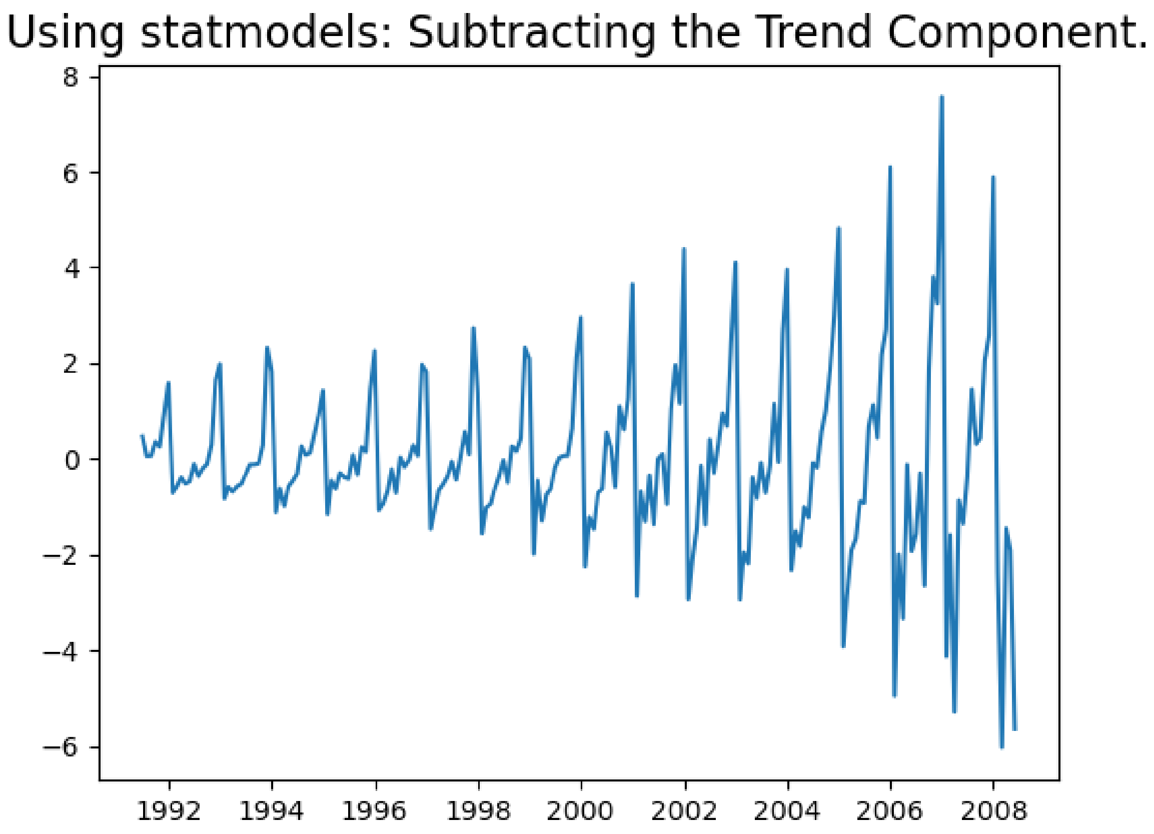
3, 减去时间序列的mean

4, 使用Baxter-King filter (statsmodels.tsa.filters.bkfilter) 或者 Hodrick-Prescott filtere (statsmodels.tsa.filters.hpfilter) 去移除 moving average trend line或者 cyclical component

```
import matplotlib.pyplot as plt
import pandas as pd

# Using scipy: Subtract the line of best fit
```

```
from scipy import signal
df=pd.read_csv('a10.csv',parse_dates=['date'])
detrended = signal.detrend(df.value.values)
plt.plot(detrended)
plt.title('Using scipy: Subtract the line of best fit',fontsize=16)
plt.show()

# Using statmodels: Subtracting the Trend Component.
from statsmodels.tsa.seasonal import seasonal_decompose
df=pd.read_csv('a10.csv',parse_dates=['date'],index_col='date')
result_mul=seasonal_decompose(df['value'],model='multiplicative',extrapolate_tre
nd='freq')
detrended=df.value.values-result_mul.trend
plt.plot(detrended)
plt.title('Using statmodels: Subtracting the Trend Component.',fontsize=16)
plt.show()
```





15， 如何去掉时间序列的季节分量

15.1 采用和seasonal window一样长的moving average， 这可以使得曲线平滑

15.2 Seasonal difference the series (subtract the value of previous season from the current value)

15.3 Divide the series by the seasonal index obtained from STL decomposition

15.3 的代码

```
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt
import pandas as pd
# Subtracting the Trend Component.
df=pd.read_csv('a10.csv',parse_dates=['date'],index_col='date')
# Time Series Decomposition
result_mul=seasonal_decompose(df['value'],model='multiplicative',extrapolate_tre
nd='freq')
# Deseasonalize
deseasonalized=df.value.values/result_mul.seasonal
# plot
plt.plot(deseasonalized)
plt.title('Drug Sale',fontsize=16)
plt.show()
```

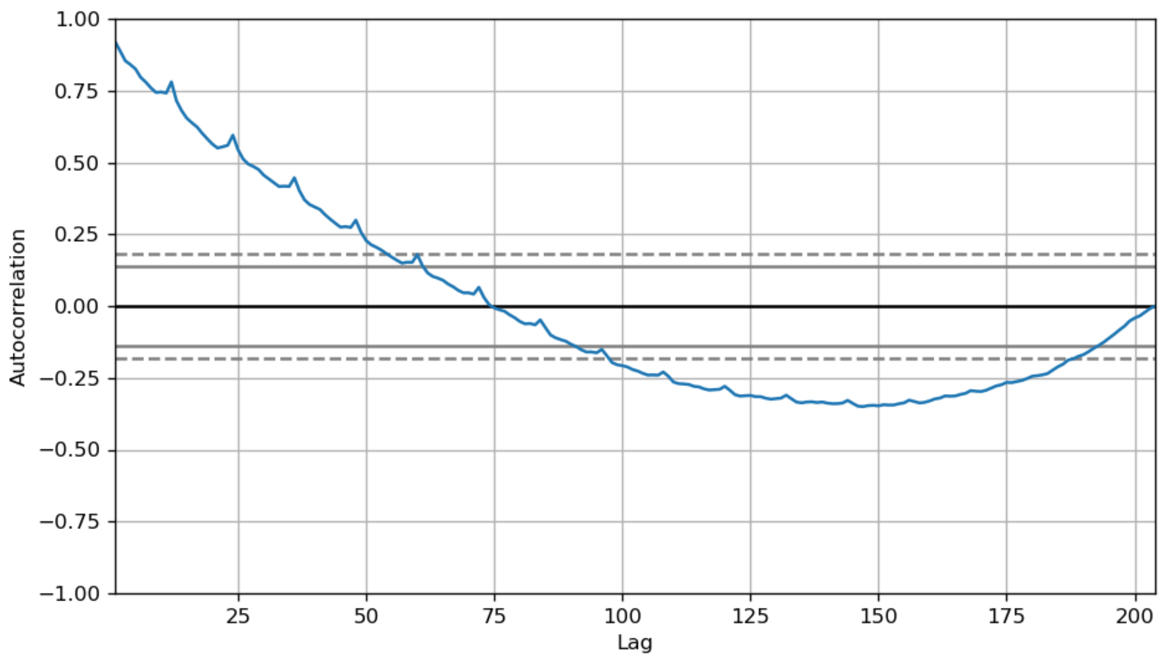
15.1 相关的说明

If dividing by the seasonal index does not work well, try taking a log of the series and then do the deseasonalizing. You can later restore to the original scale by taking an exponential.

16, 如何测试一个时间序列的seasonality

最常见的方式就是画出这个时间序列，然后检查在固定时间区间里面的重复的模式，所以这种类型的seasonality就是由clock或者calender决定的。 更加确定的方式就是使用Autocorrelation fucntion (AF)plot.

```
import matplotlib.pyplot as plt
import pandas as pd
from pandas.plotting import autocorrelation_plot
df=pd.read_csv('a10.csv')

plt.rcParams.update({'figure.figsize':(9,5),'figure.dpi':120})
autocorrelation_plot(df.value.tolist())
plt.show()
```



16.1 相关的说明

I must caution you that in real word datasets such strong patterns is hardly noticed and can get distorted by any noise, so you need a careful eye to capture these patterns.

Alternately, if you want a statistical test, the CHTest can determine if seasonal differencing is required to stationarize the series.

17, 如何对待时间序列里面缺失的值

有可能是这个数据在这些时间比你更不是有效的，这时候这些测量可能是0，这些情况下你可以能把这些时间段填为0

2,我们不应该用时间序列的平均值来填满，尤其是当这个序列是非平稳的，

3，依赖于时间序列的特点，可以使用不同的方法，多尝试

方法如下：

1， backward Fill

2, Linear interpolation

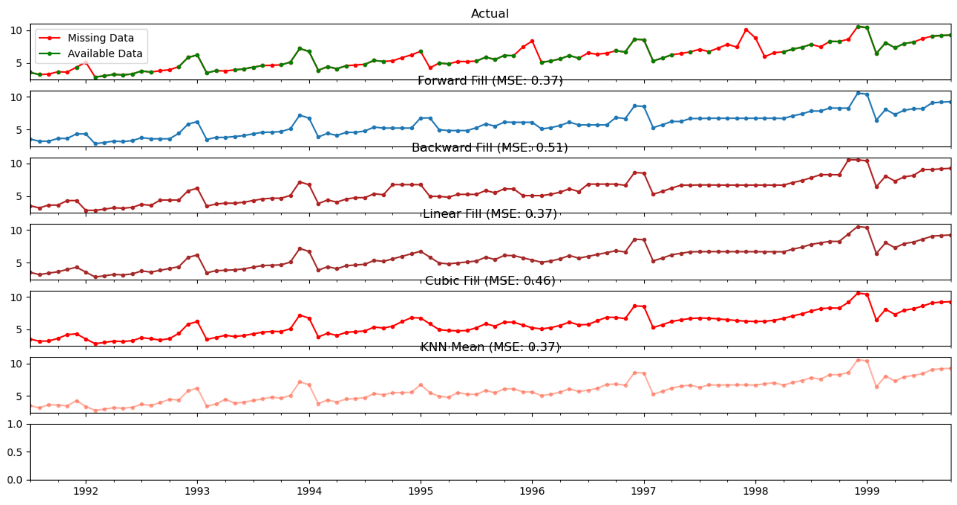
3, Quadratic interpolation

4, Mean of nearest neighbors

5, Mean of seasonal couterparts

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from pandas.plotting import autocorrelation_plot

# # Generate dataset
from scipy.interpolate import interp1d
from sklearn.metrics import mean_squared_error
df_orig = pd.read_csv('a10.csv', parse_dates=['date'],
index_col='date').head(100)
df = pd.read_csv('a10_missings.csv', parse_dates=['date'], index_col='date')

fig, axes = plt.subplots(7, 1, sharex=True, figsize=(10, 12))
plt.rcParams.update({'xtick.bottom' : False})

## 1. Actual -------------------------------
df_orig.plot(title='Actual', ax=axes[0], label='Actual', color='red', style=".-
")
df.plot(title='Actual', ax=axes[0], label='Actual', color='green', style=".-")
axes[0].legend(["Missing Data", "Available Data"])


## 2. Forward Fill --------------------------
# df 是缺失一些值得
df_ffill = df.ffill()
error = np.round(mean_squared_error(df_orig['value'], df_ffill['value']), 2)
df_ffill['value'].plot(title='Forward Fill (MSE: ' + str(error) +")",
ax=axes[1], label='Forward Fill', style=".-")


## 3. Backward Fill -------------------------
df_bfill = df.bfill()
error = np.round(mean_squared_error(df_orig['value'], df_bfill['value']), 2)
df_bfill['value'].plot(title="Backward Fill (MSE: " + str(error) +")",
ax=axes[2], label='Back Fill', color='firebrick', style=".-")


## 4. Linear Interpolation ------------------
df['rownum'] = np.arange(df.shape[0])
df_nona = df.dropna(subset = ['value'])
f = interp1d(df_nona['rownum'], df_nona['value'])
df['linear_fill'] = f(df['rownum'])
error = np.round(mean_squared_error(df_orig['value'], df['linear_fill']), 2)
df['linear_fill'].plot(title="Linear Fill (MSE: " + str(error) +")", ax=axes[3],
label='Cubic Fill', color='brown', style=".-")


## 5. Cubic Interpolation --------------------
f2 = interp1d(df_nona['rownum'], df_nona['value'], kind='cubic')
df['cubic_fill'] = f2(df['rownum'])
error = np.round(mean_squared_error(df_orig['value'], df['cubic_fill']), 2)
```

```
df['cubic_fill'].plot(title="Cubic Fill (MSE: " + str(error) +")", ax=axes[4],
label='Cubic Fill', color='red', style=".-")


# Interpolation References:
# https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html
# https://docs.scipy.org/doc/scipy/reference/interpolate.html

## 6. Mean of 'n' Nearest Past Neighbors ------
def knn_mean(ts, n):
    out = np.copy(ts)
    for i, val in enumerate(ts):
        if np.isnan(val):
            n_by_2 = np.ceil(n/2)
            lower = np.max([0, int(i-n_by_2)])
            upper = np.min([len(ts)+1, int(i+n_by_2)])
            ts_near = np.concatenate([ts[lower:i], ts[i:upper]])
            out[i] = np.nanmean(ts_near)
    return out

df['knn_mean'] = knn_mean(df.value.values, 8)
error = np.round(mean_squared_error(df_orig['value'], df['knn_mean']), 2)
df['knn_mean'].plot(title="KNN Mean (MSE: " + str(error) +")", ax=axes[5],
label='KNN Mean', color='tomato', alpha=0.5, style=".-")
plt.show()
```



17.1 相关的说明

You could also consider the following approaches depending on how accurate you want the imputations to be.

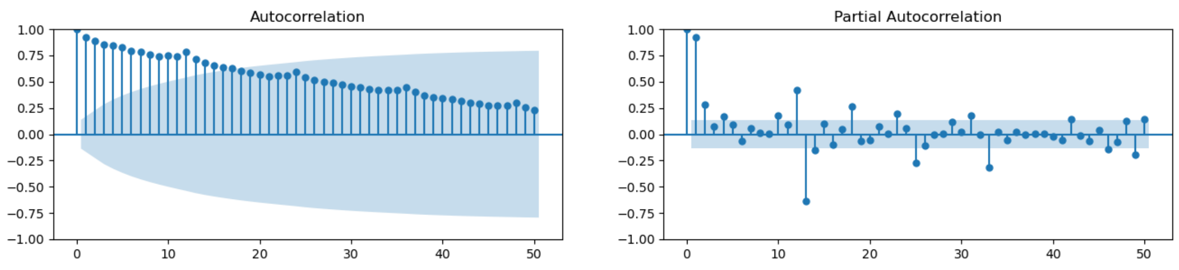
1. If you have explanatory variables use a prediction model like the random forest or k-Nearest Neighbors to predict it.
2. If you have enough past observations, forecast the missing values.
3. If you have enough future observations, backcast the missing values
4. Forecast of counterparts from previous cycles.

18, 如何画autocorrelation and partial autocorrelation functions

自相关是一个序列和它自己的lag,如果一个序列是significantly autocorrelated， 则这个序列、lag之前的值有助于预测现在的值

partial autocorrelation相似，但是只传递一个序列和它的lag的相关性，而没有和中间的lag的相关性

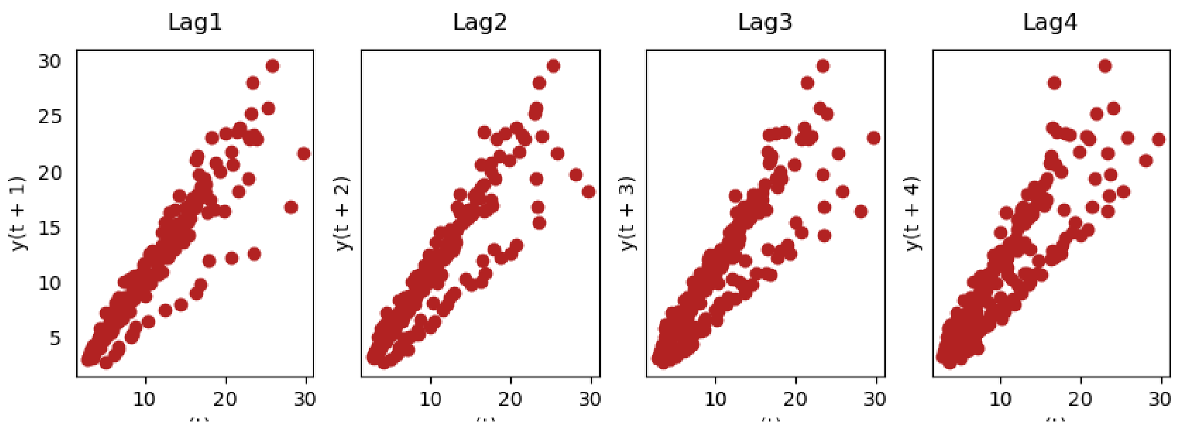
```
import matplotlib.pyplot as plt
import pandas as pd
from pandas.plotting import autocorrelation_plot

from statsmodels.tsa.stattools import acf,pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

df=pd.read_csv('a10.csv')
fig,axes=plt.subplots(1,2,figsize=(16,3),dpi=100)
plot_acf(df.value.tolist(),lags=50,ax=axes[0])
plot_pacf(df.value.tolist(),lags=50,ax=axes[1])
plt.show()
```



19， 如何画出lag plot用于检查autocorrelation

```
from statsmodels.tsa.stattools import acf,pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

df=pd.read_csv('a10.csv')
fig,axes=plt.subplots(1,2,figsize=(16,3),dpi=100)
plot_acf(df.value.tolist(),lags=50,ax=axes[0])
plot_pacf(df.value.tolist(),lags=50,ax=axes[1])

from pandas.plotting import lag_plot
plt.rcParams.update({'ytick.left': False, 'axes.titlepad':10})

# import
ss=pd.read_csv('sunspotarea.csv')
a10=pd.read_csv('a10.csv')

# plot
fig, axes=plt.subplots(1,4,figsize=(10,3),sharex=True,sharey=True,dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(ss.value,lag=i+1,ax=ax,c='firebrick')
    ax.set_title('Lag'+str(i+1))
fig.suptitle('Lag')

fig,axes=plt.subplots(1,4,figsize=(10,3),sharex=True,sharey=True,dpi=100)
```

```
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(a10.value,lag=i+1,ax=ax,c='firebrick')
    ax.set_title('Lag'+str(i+1))

fig.suptitle('Lag',y=1.05)
```



20， approximate、sample entroy用于评估序列的可预测性

数值越高，越难预测

sample entropy是对于大小序列的评估都很一致

```
import pandas as pd
import numpy as np


# import
ss=pd.read_csv('sunspotarea.csv')
a10=pd.read_csv('a10.csv')

rand_small=np.random.randint(0,100,size=36)
rand_big=np.random.randint(0,100,size=136)

def ApEn(U,m,r):
    def _maxdist(x_i,x_j):
        return max([abs(ua-va) for ua,va in zip(x_i, x_j)])

    def _phi(m):
        x=[[U[j] for j in range(i,i+m-1+1)] for i in range(N-m+1)]
        C=[len([1 for x_j in x if _maxdist(x_i,x_j)<=r])/(N-m+1.0) for x_i in x]
        return (N-m+1.0)**(-1)*sum(np.log(C))

    N=len(U)
    return abs(_phi(m+1)-_phi(m))

print(ApEn(ss.value,m=2,r=0.2*np.std(ss.value))) #0.651
print(ApEn(a10.value,m=2,r=0.2*np.std(a10.value))) #0.537
print(ApEn(rand_small,m=2,r=0.2*np.std(rand_small))) #0.143
print(ApEn(rand_big,m=2,r=0.2*np.std(rand_big))) #0.716

## https: //en.wikipedia.org
def SampEn(U,m,r):
    """Compute Sample Entropy"""
    def _maxdist(x_i,x_j):
        return max([abs(ua-va) for ua, va in zip(x_i, x_j)])

    def _phi(m):
        x=[[U[j] for j in range(i,i+m-1+1)] for i in range(N-m+1)]
        C=[len([1 for j in range(len(x)) if i!=j and _maxdist(x[i],x[j])<=r])
for i in range(len(x))]
        return sum(C)

    N=len(U)
    return -np.log(_phi(m+1)/_phi(m))

print(SampEn(ss.value,m=2,r=0.2*np.std(ss.value)))
print(SampEn(a10.value,m=2,r=0.2*np.std(a10.value)))
print(SampEn(rand_small,m=2,r=0.2*np.std(rand_small)))
print(SampEn(rand_big,m=2,r=0.2*np.std(rand_big)))
```

```
0.6514704970333534
0.5374775224973489
0.16788956147066747
0.6974993314580624
0.7853311366380039
0.41887013457621214
1.791759469228055
2.5649493574615367
```

20.1 相关的说明

The more regular and repeatable patterns a time series has, the easier it is to forecast. The 'Approximate Entropy' can be used to quantify the regularity and unpredictability of fluctuations in a time series.

Sample Entropy is similar to approximate entropy but is more consistent in estimating the complexity even for smaller time series. For example, a random time series with fewer data points can have a lower 'approximate entropy' than a more 'regular' time series, whereas, a longer random time series will have a higher 'approximate entropy'.

21，如何平滑曲线以及为什么平滑曲线

平滑时间序列对于如下是有用的：

- Reducing the effect of noise in a signal get a fair approximation of the noise-filtered series.
- The smoothed version of series can be used as a feature to explain the original series itself.
- Visualize the underlying trend better

如何平滑一个时间序列呢?

1. Take a moving average
2. Do a LOESS smoothing (Localized Regression)
3. Do a LOWESS smoothing (Locally Weighted Regression)

22，Granger Causality test去表明一个时间序列是否对于预测另一个时间序列有帮助

一定要清楚地知道每种方法的优点和缺点

pytorch dataloader的用法：一个简单的介绍

如何使用**ipynb**文件

现代信号分析的一些基础处理

参考：哥延根数学学派,知乎

pytorch的基础使用说明

参考：*Dive into deep learning*

参考：https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

如何写出简洁易懂的**python**代码

参考：*effective python*

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.

Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

机器学习调参的基础方法

参考论文：https://www.zhihu.com/question/285071001/answer/2795849067

机器学习常见工程上的提醒

参考书籍：

机器学习中如何设置随机性保证复现

强化学习使用pytorch实现

常见的时序数据集

sea

时间序列的特征工程

Jupyter notebook的使用方法

时间序列常见的的损失函数

参考论文：*A Comprehensive Survey of Regression Based Loss Functions for Time Series Forecasting*

1, Mean Absolute Error

2, Mean Square Error

3, Mean Bias Error (MBE)

4, Relative Absolute Error (RAE)

where

5, Relative Squared Error (RSE)

where

6, Mean Absolute Percentage Error (MAPE)

7, Root Mean Squared Error (RMSE)

8, Mean Squared Logarithmic Error (MSLE)

9, Root Mean Squared Logarithmic Error (RMSLE)

10, Normalized Root Mean Squared Error (NRMSE)

where

常见的优化器的比较

常见的机器学习优化算法及其实现

时序预测和风力预测有哪些表现的形式

数据增强 (Data augumentation)

参考： *Time Series Data Augmentation for Deep Learning: A Survey*

linux cmd的一些基础命令

时序预测相关的论文

1 传统方向

2 纯深度学习方向

2.1 LSTM用于时间序列

代码： A minimal implementation

2.2 RNN用于时间序列预测

代码： A minimal implementation

参考：https://github.com/Azure/DeepLearningForTimeSeriesForecasting

2.3 CNN用于时间序列预测

2.4 BiLSTM用于时间序列预测

ARGU用于时间序列预测

MLP用于时间序列预测

TCN用于时间序列预测

参考书籍： *Time Series Forecasting using Deep Learning: Combining PyTorch, RNN, TCN, and Deep Neural Network Models to Provide Production-Ready Prediction Solutions*

GRU (Gated Recurrent Unit) 用于时间序列预测(pytorch 实现)

Autoregression (AR)

参考：https://github.com/jiwidi/time-series-forecasting-with-python

Moving Average

Autoregressive Moving Average(ARMA)

Autoregressive integraded moving average (ARIMA)

Seasonal autoregressive integrated moving average (SARIMA)

Bayesian regression

Lasso

SVM

Random forest

Nearest neighbors

XGBoost

Lightgbm

Prophet

DeepAR

概率时间序列建模

参考： https://github.com/awslabs/gluonts

2.5 Autoinformer用于时间序列预测

2.6 Informer 用于时间序列预测

2.7 Pyraformer

2.8 LogTrans

2.9 Reformer

2.10 LSTNet

3 信号处理+ 深度学习方向

4 纯信号处理方向

5，多模态融合的时间序列预测（博士方向）

时间序列预测网络有哪些可以用来比较的性能指标

1， 算法复杂度

2，

风机功率预测有哪些自己的特点

时空融合的风力功率预测

自创的可以体现性能的指标

AutoML自动调整参数方法

强化学习用于调整参数

Robust 时间序列预测

参考论文：*Robust Time Series Analysis and Applications: An Industrial Perspective*, in *KDD* 2022

如何使用Cuda自定义算子

一篇论文最基本的内容

参考文献：

开源的代码：

相关领域研究的综述？

1.1 论文试图解决什么问题？

1.1.1 可以提供一个具体的例子嘛

1.2 这是否是一个新问题

1.3 这篇文章要验证一个什么科学假设

1.4 有哪些相关研究，如何分类？ 谁是这一课题在领域内值得关注的研究员

1.4.1 请对别人的工作进行comment

1.4.2 请画出创新表

1.4.3 请从假设，解决的问题，之前工作的不足，没有研究过的一些gap，你的方法的优点（时间，空间，内存复杂度，更加精准), 阐述你的方法与别人的方法的不足

1.4.4 请画出相关领域研究的timeline

1.5 论文中提到的解决方案的关键是什么

1.5.1 请用一张图画出你解决方案的核心

1.5.2 可以提供一个具体的例子吗

1.5.3 相关的阐述所出现的数学符号

1.5.4 这个方法在物理和直观上的相关解释

1.5.5 这个方法同物理定律相结合的点

1.6 论文中的实验是如何设计的？

1.7 用于定量评估的数据集是什么？ 代码有没有开源

1.7.1 请用定性的数据描述数据集的不同

1.7.2 请从感性的角度描述数据集的不同

1.8 论文中的实验及结果有没有很好地支持需要验证的科学假设？

1.8.1 实验设计性能指标有哪些？

1.8.2 为什么选取这些性能指标？ 这些性能指标有哪些优劣？

1.8.3 实验设计选取的**benchmark**有哪些？各种**benchmark**的优劣？

1.8.3 这些**benchmark**的选取的原因？ 是否具有代表性

1.8.4 实验超参数的数量和选取

1.9 这篇论文到底有什么贡献？

1.9.1 这种方法有什么缺点或者不适用的地方

1.9.2 哪些方法是会对别人有益的？ **take-home message**有哪些？

1.10 下一步呢？ 有什么工作可以继续深入？

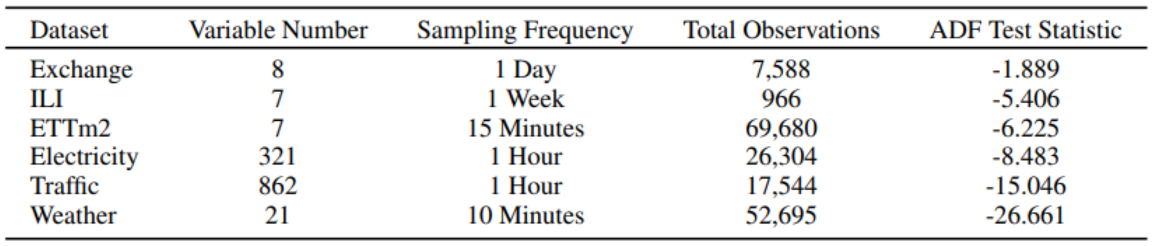
##

项目**3: Empirical mode decomposition based** 时间序列预测

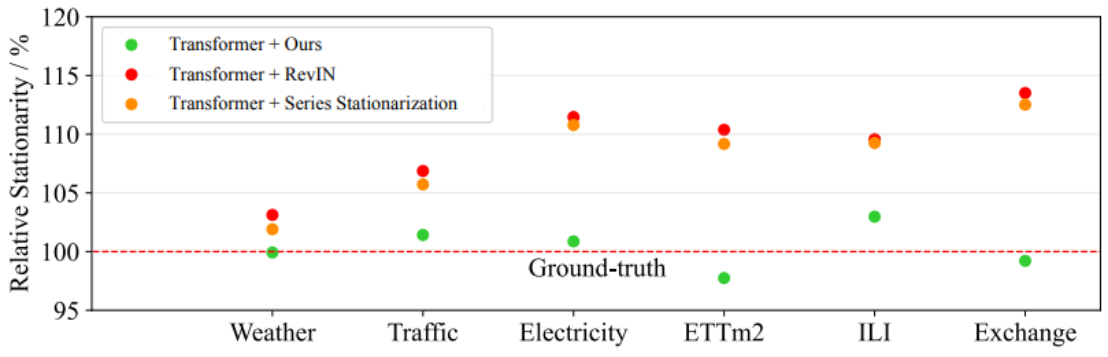
文章有哪些比较好的表现形式

数据集

Table 1: Summary of datasets. Smaller ADF test statistic indicates more stationary dataset.



Dataset	Variable Number	Sampling Frequency	Total Observations	ADF Test Statistic
Exchange	8	1 Day	7,588	-1.889
ILI	7	1 Week	966	-5.406
ETTm2	7	15 Minutes	69,680	-6.225
Electricity	321	1 Hour	26,304	-8.483
Traffic	862	1 Hour	17,544	-15.046
Weather	21	10 Minutes	52,695	-26.661



一些常用的工具

Docker

Conda

Weights & Biases

MLFlow

Screen

GitHub

Lucidchart

Streamlit

高引用论文的特点

1, 论文的引用和数字强相关,所以主要写长文，而不是短文，去吸引论文

2， 多个作者更容易引起曝光，每个作者都有自己的社交网络，多个作者也可以自己引用，而且多个作者可以改进论文的质量

3，图表和表格越多，可以传递更多信息，如果论文的图表有限制，则把多个图合并成一个图可以是一个好主意，至少要6张图，平均的是七张图，但是nature是5张或者六张图

4，表格的数量越多越好，但是要避免冗余的表格，至少要两张表格，平均是三张表格

5，在多个网址发表自己的论文，researchgate, google scholar, Facebook, Twitter， Linkdin, 等多个地方推广自己的论文

6，尽可能地包含尽可能多的公式，没必要计算公式，但是如果需要的话就大量使用

7，写好的title对于吸引读者很重要

8，和高引被作者的交流，高引被的作者不一定互相合作，但是会大量的互相引用，

9，标题的问号和/，但是colon,dash， dot很重要，标题大概10+-3，

10，包含6个或者更多的作者

11，35000个字符，不包含空格和参考文献

如何做presentation

1, 一个月之前就做slide， 然后给合作者提供足够多的时间修改，不要在最后一天开始做

2，不要直接丢图，如果是一篇论文的话，思考presentation中最核心的部分，关注最主要的信息并且提供相应的support,如果有视屏或者图，加入这些东西

3，最小的做presentation的时间是两个小时，一个slide大概有1-1.5分钟，实际讲一下测试一下时间，如果有很多材料，不要试图讲的太快，关注主要信息，放弃次要信息，如果presentation太短，则检查topic，扩充它

4，发给coauthor and supervisor 去检查，是否同意你的观点，是否有什么建议，给与他们一到两周，然后返修回来，给30min-2h的时间修改

5, 实地演练：1，对于讲故事的方式足够自信，2,在允许的时间完成，至少测试三次，第一次是时间，第二次是为了确认内容，第三次是为了在开始前的一夜完成所有材料的浏览

6，内部的展示，和supervisors和一些博士同事讨论他们是否对你的内容感兴趣，可以组织一次meeting, 安排好房间，咖啡，甚至午饭，留一部分时间来反馈和讨论

7，了解你的听众的背景，如果你的听众不是小同行，你可能需要warm up your message, 为什么你的研究重要，然后那些点需要被提及去链接你的工作，

8，选择一个take-home message, 多次重复，或者单独放一页（用更大的字体）

常见的presentation的一些语言

1，overview slide (第一部分)

“Today I want to convince you of My Key Message. For that reason, I will cover the following topics:… “

2，A clear introduction

区分你听众的背景，关注不同的方面,花一点时间阐述你的研究工作对于他们的研究工作有一定的影响，描述一个更大的问题你需要通过你的研究去解决，不要直接进入到你的研究内容以及所有的内容。 如果你的研究是更大项目的一个部分，可以和项目的其他人一起讨论，介绍其他的研究项目，并且写出他们的名字

比如：内部presentation，则可以跳过背景和literature review,花更多的时间在细节上面

比如：是实际的人员，则需要侧重你的研究如何影响他们日常的工作

比如：基金委，则需要说明你研究的价值以及未来资助的需要

比如：会议，则都可以，关注细节或者关注一些意义

3，不要涉及多一点的公式

如果一个公式真的很重要，写下来，花足够多的时间解释所有的变量，圈出变量，然后一步一步去解释，如果只考虑两个变量的关系，使用下面这句话：

What I want you to notice here is the relation between This Parameter and Result of Equation.”

也可以用动画去圈出这个参数去提醒听众

4，解释图的每一个组成，不要直接跳过

比如x-axis的变量及它的单位，y-axis及它的单位，数据点，是否是原始数据，或者经过中间计算，解释legend, 然后才描述你的结论

5， 不要最后只有你的总结和结论的slide

如果有 Q & A的时间，可以使用

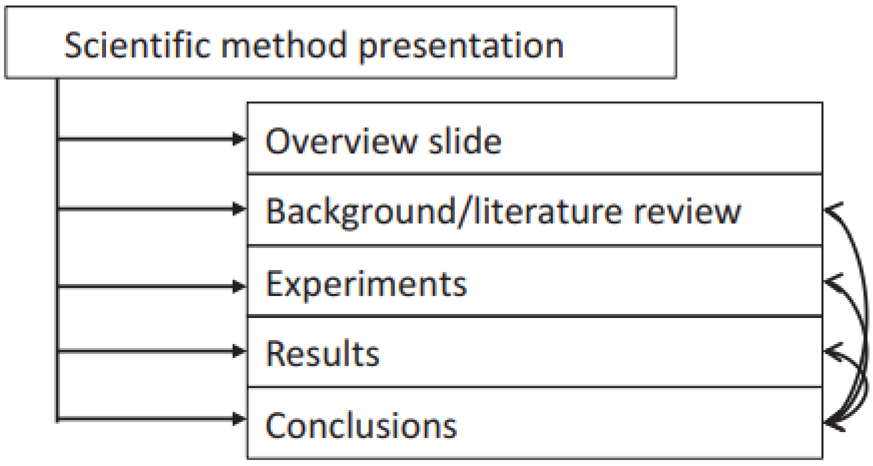
“That concludes my presentation for today, and I’d welcome any questions you may have

如果没有Q & A的时间，可以使用：

That concludes my presentation for today, and I'd like to thank you for your attention,"

最后一页可以用thank you 结尾

做presentation的第一种结构



Background/Literature Review

不要简单的列举参考文献，而应该关注你的研究课题为什么重要，然后把它扩大一点，然后展示有哪些工作已经做了，尽可能用图和照片来展示，显示和你的研究直接相关的部分。 除非你的工作是literature review， 否则重点还是展示你自己的工作

ExperIment

解释你做了什么，然后是尽可能用visual, 如果有video，用video。 可以使用timelapse（隔一段时间拍一下），如果你的实验数据是数字的，则可以使用软件package里面的一些工具去展示你的工作

Result

提及你的实验结果，关注主要的结果，主要是展示key message的support, 如果可以的话，尽量用图

Conclusion

和之前的所有相关的链接起来

做presentation的第二种结构

Overview

Recommendations/hypothesis

提出假设和自己的key message, 然后一步一步验证它

Tools used to prove hypothesis

展示需要用来证明自己key message的工具， 可以用文献中找你使用过的材料，比如数据库你用来比较结果的那个

Actual Proof

这一部分占了绝大部分，一步一步给他们展示

consequence

展示这个东西更大的意义，和你的听众的实际相结合，然后展示你的工作对于他们的实际工作是有意义的，展示你的结果和真实案例结合的样子，如果案例研究不是学术性的，重新考虑找一个案例，没有没有case study,则可以解释你的工作如何影响你的领域。

可以用：

after the proof, I will show you how these recommendations can be applied to your daily practice”

outlook

告诉你的听众哪些问题依然是open的，以及你最近在做哪些工作。

Q & A环节

1，如果有问题，这是好事，无论是现场还是一对一，如果没问题，则说明可能没人关心你的结果，需要反思自己的presentation的方式

2，留充分的时间去解答问题，如果在现场，首先说一下问题，然后才解答。如果你认为问题无关，不要很快就跳过，解释一下为什么应该关注其他的方面，尊重对你研究感兴趣的人。不要打断对方，然后回答，至少应该把问题听完。

语言问题

1，不要带小抄直接读，这样非常不礼貌

2，不要把所有的东西都放在slide上面，这超级无聊

3，不要不做slide就展示，这样可能会导致他们的presentation非常假

4，不要不准备回答的问题

应当做的

1，多联系口语

2，有意识地联系你口语的弱点，每天都听广播

3，面对你的问题，解决它

4，沉没在英文环境当中

展示之后应该做什么

1，添加到个人主页里面的talks

2, 把slide放在网上，注意没有在slide里面没有敏感信息，然后你有权去分享你的slide, 如果有疑问，问自己的导师和资助方。如果都可以的话，把东西放在Slideshare这个网站，这个网站的东西可以直接分享在 Linkedin, Twitter, Facebook, an internet forum, email, 或者在博客上分享

如何写博客：展示自己的研究