01.导入数据

```
# 从 CSV 文件导入数据
pd.read_csv('file.csv', name=['列名','列名2'])
# 从限定分隔符的文本文件导入数据
pd.read_table(filename, header=0)
# Excel 导入，指定 sheet 和表头
pd.read_excel('file.xlsx', sheet_name=' 表1', header=0)
# 从 SQL 表/库导入数据
pd.read_sql(query, connection_object)
# 从 JSON 格式的字符串导入数据
pd.read_json(json_string)
# 解析 URL、字符串或者 HTML 文件，抽取其中的 tables 表格
pd.read_html(url)
# 从你的粘贴板获取内容，并传给 read_table()
pd.read_clipboard()
# 从字典对象导入数据，Key 是列名，Value是数据
pd.DataFrame(dict)
# 导入字符串
from io import StringIO
pd.read_csv(StringIO(web_data.text))
```

02.导出数据

```
# 导出数据到CSV文件
df.to_csv('filename.csv')
# 导出数据到Excel文件
df.to_excel('filename.xlsx', index=True)
# 导出数据到 SQL 表
df.to_sql(table_name, connection_object)
# 以Json格式导出数据到文本文件
df.to_json(filename)
# 其他
df.to_html()  # 显示 HTML 代码
df.to_markdown() # 显示 markdown 代码
df.to_string() # 显示格式化字符
df.to_latex(index=False) # LaTeX tabular, longtable
df.to_dict('split') # 字典，格式 list/series/records/index
df.to_clipboard(sep=',', index=False) # 存入系统剪贴板

# 将两个表格输出到一个excel文件里面,导出到多个 sheet
writer=pd.ExcelWriter('new.xlsx')
df_1.to_excel(writer,sheet_name='第一个', index=False)
df_2.to_excel(writer,sheet_name='第二个', index=False)
```

```
writer.save() # 必须运行writer.save()，不然不能输出到本地

# 写法2
with pd.ExcelWriter('new.xlsx') as writer:
    df1.to_excel(writer, sheet_name='第一个')
    df2.to_excel(writer, sheet_name='第二个')
```

03.创建测试对象

```
# 创建20行5列的随机数组成的 DataFrame 对象
pd.DataFrame(np.random.rand(20,5))

# 从可迭代对象 my_list 创建一个 Series 对象
pd.Series(my_list)

# 增加一个日期索引
df.index = pd.date_range('1900/1/30', periods=df.shape[0])

# 创建随机数据集
df = pd.util.testing.makeDataFrame()

# 创建随机日期索引数据集
df = pd.util.testing.makePeriodFrame()
df = pd.util.testing.makeTimeDataFrame()

# 创建随机混合类型数据集
df = pd.util.testing.makeMixedDataFrame()
```

04.查看、检查、统计、属性

```
df.head(n) # 查看 DataFrame 对象的前n行
df.tail(n) # 查看 DataFrame 对象的最后n行
df.sample(n) # 查看 n 个样本，随机
df.shape # 查看行数和列数
df.info() # 查看索引、数据类型和内存信息
df.describe() # 查看数值型列的汇总统计
df.dtypes # 查看各字段类型
df.axes # 显示数据行和列名
pd.isnull() # 检查DataFrame对象中的空值，并返回一个 Boolean 数组
pd.notnull() # 检查DataFrame对象中的非空值，并返回一个 Boolean 数组

df.mean() # 返回所有列的均值
df.mean(1) # 返回所有行的均值，下同
df.corr() # 返回列与列之间的相关系数
df.count() # 返回每一列中的非空值的个数
```

```
df.max() # 返回每一列的最大值
df.min() # 返回每一列的最小值
df.median() # 返回每一列的中位数
df.std() # 返回每一列的标准差
df.var() # 方差
s.mode() # 众数
s.prod() # 连乘
s.cumprod() # 累积连乘,累乘
df.cumsum(axis=0) # 累积连加,累加
s.nunique() # 去重数量，不同值的量
df.idxmax() # 每列最大的值的索引名
df.idxmin() # 最小

df.columns # 显示所有列名
df.team.unique() # 显示列中的不重复值
# 查看 Series 对象的唯一值和计数，计数占比：normalize=True
s.value_counts(dropna=False)
# 查看 DataFrame 对象中每一列的唯一值和计数
df.apply(pd.Series.value_counts)

df.duplicated() # 重复行
df.drop_duplicates() # 删除重复行
# set_option、reset_option、describe_option 设置显示要求
pd.get_option()

# 设置行列最大显示数量，None 为不限制
pd.options.display.max_rows = None
pd.options.display.max_columns = None
df.col.argmin() # 最大值[最小值 .argmax()] 所在位置的自动索引
df.col.idxmin() # 最大值[最小值 .idxmax()] 所在位置的定义索引

# 累计统计
ds.cumsum() # 前边所有值之和
ds.cumprod() # 前边所有值之积
ds.cummax() # 前边所有值的最大值
ds.cummin() # 前边所有值的最小值

# 窗口计算(滚动计算)
ds.rolling(x).sum() #依次计算相邻x个元素的和
ds.rolling(x).mean() #依次计算相邻x个元素的算术平均
ds.rolling(x).var() #依次计算相邻x个元素的方差
ds.rolling(x).std() #依次计算相邻x个元素的标准差
ds.rolling(x).min() #依次计算相邻x个元素的最小值
ds.rolling(x).max() #依次计算相邻x个元素的最大值
```

05.数据清理

```
df.columns = ['a','b','c'] # 重命名列名
df.columns = df.columns.str.replace(' ', '_') # 列名空格换下划线
df.loc[df.AAA >= 5, ['BBB', 'CCC']] = 555 # 替换数据
df['pf'] = df.site_id.map({2: '小程序', 7:'M 站'}) # 将枚举换成名称

pd.isnull() # 检查DataFrame对象中的空值，并返回一个 Boolean 数组
pd.notnull() # 检查DataFrame对象中的非空值，并返回一个 Boolean 数组

df.drop(['name'], axis=1) # 删除列
df.drop([0, 10], axis=0) # 删除行
del df['name'] # 删除列
df.dropna() # 删除所有包含空值的行
df.dropna(axis=1) # 删除所有包含空值的列
df.dropna(axis=1,thresh=n) # 删除所有小于 n 个非空值的行
df.fillna(x) # 用x替换DataFrame对象中所有的空值
df.fillna(value={'prov':'未知'}) # 指定列的空值替换为指定内容

s.astype(float) # 将Series中的数据类型更改为 float 类型
df.index.astype('datetime64[ns]') # 转化为时间格式

s.replace(1, 'one') # 用 ‘one’ 代替所有等于 1 的值
s.replace([1, 3],['one','three']) # 用'one'代替 1，用 'three' 代替 3

df.rename(columns=lambda x: x + 1) # 批量更改列名
df.rename(columns={'old_name': 'new_name'}) # 选择性更改列名
df.set_index('column_one') # 更改索引列
df.rename(index=lambda x: x + 1) # 批量重命名索引

# 重新命名表头名称
df.columns = ['UID', '当前待打款金额', '认证姓名']
df['是否设置提现账号'] = df['状态'] # 复制一列
df.loc[:, ::-1] # 列顺序反转
df.loc[::-1] # 行顺序反转，下方为重新定义索引
df.loc[::-1].reset_index(drop=True)
```

06.数据处理：Filter、Sort

```
# 保留小数位，四舍六入五成双
df.round(2) # 全部
df.round({'A': 1, 'C': 2}) # 指定列
df['Name'] = df.Name # 取列名的两个方法
df[df.index == 'Jude'] # 索引列的查询要用 .index
df[df[col] > 0.5] # 选择col列的值大于0.5的行

# 多条件查询
df[(df['team'] == 'A') &
   ( df['Q1'] > 80) &
```

```
   df.utype.isin(['老客', '老访客'])]

# 筛选为空的内容
df[df.order.isnull()]
# 类似 SQL where in
df[df.team.isin('A','B')]
df[(df.team=='B') & (df.Q1 == 17)]
df[~(df['team'] == 'A') | ( df['Q1'] > 80)] # 非，或
df[df.Name.str.contains('张')] # 包含字符

# 排序
df.sort_values(col1) # 按照列col1排序数据，默认升序排列
df.col1.sort_values() # 同上，-> s
df.sort_values(col2, ascending=False) # 按照列 col1 降序排列数据
# 先按列col1升序排列，后按col2降序排列数据
df.sort_values([col1,col2], ascending=[True,False])
df2 = pd.get_dummies(df, prefix='t_') # 将枚举的那些列带枚举转到列上
s.set_index().plot()

# 多索引处理
dd.set_index(['utype', 'site_id', 'p_day'], inplace=True)
dd.sort_index(inplace=True) # 按索引排序
dd.loc['新访客', 2, '2019-06-22'].plot.barh() # loc 中按顺序指定索引内容
# 前100行，不能指定行，如：df[100]
df[:100]
# 只取指定行
df1 = df.loc[0:, ['设计师ID', '姓名']]
# 将ages平分成5个区间并指定 labels
ages = np.array([1,5,10,40,36,12,58,62,77,89,100,18,20,25,30,32])
pd.cut(ages, [0,5,20,30,50,100],
       labels=[u"婴儿",u"青年",u"中年",u"壮年",u"老年"])

daily_index.difference(df_work_day.index) # 取出差别

# 格式化
df.index.name # 索引的名称 str
df.columns.tolist()
df.values.tolist()
df.总人口.values.tolist()
data.apply(np.mean) # 对 DataFrame 中的每一列应用函数 np.mean
data.apply(np.max,axis=1) # 对 DataFrame 中的每一行应用函数 np.max
df.insert(1, 'three', 12, allow_duplicates=False) # 插入列（位置、列名、[值]）
df.pop('class') # 删除列

# 增加一行
df.append(pd.DataFrame({'one':2,
                        'two':3,
                        'three': 4.4},
                       index=['f']),
          sort=True)
```

```python
# 指定新列
iris.assign(sepal_ratio=iris['SepalWidth'] / iris['SepalLength']).head()
df.assign(rate=lambda df: df.orders/df.uv)

# shift 函数是对数据进行平移动的操作
df['增幅'] = df['国内生产总值'] - df['国内生产总值'].shift(-1)
df.tshift(1) # 时间移动，按周期
# 和上相同，diff 函数是用来将数据进行移动之后与原数据差
# 异数据，等于 df.shift()-df
df['增幅'] = df['国内生产总值'].diff(-1)
# 留存数据，因为最大一般为数据池
df.apply(lambda x: x/x.max(), axis=1)

# 取 best 列中值为列名的值写到 name 行上
df['value'] = df.lookup(df['name'], df['best'])

s.where(s > 1, 10) # 满足条件下数据替换（10，空为 NaN）
s.mask(s > 0) # 留下满足条件的，其他的默认为 NaN
# 所有值加 1（加减乘除等）
df + 1 / df.add(1)

# 管道方法，链式调用函数，f(df)=df.pipe(f)
def gb(df, by):
    result = df.copy()
    result = result.groupby(by).sum()
    return result

# 调用
df.pipe(gb, by='team')
# 窗口计算 '2s' 为两秒
df.rolling(2).sum()
# 在窗口结果基础上的窗口计算
df.expanding(2).sum()
# 超出（大于、小于）的值替换成对应值
df.clip(-4, 6)
# AB 两列想加增加 C 列
df['C'] = df.eval('A+B')
# 和上相同效果
df.eval('C = A + B', inplace=True)
# 数列的变化百分比
s.pct_change(periods=2)
# 分位数，可实现时间的中间点
df.quantile(.5)
# 排名 average，min,max,first，dense，默认 average
s.rank()
# 数据爆炸，将本列的类列表数据和其他列的数据展开铺开
df.explode('A')
# 枚举更新
status = {0:'未执行', 1:'执行中', 2:'执行完毕', 3:'执行异常'}
df['taskStatus'] = df['taskStatus'].apply(status.get)
df.assign(金额=0) # 新增字段
```

```
df.loc[('bar', 'two'), 'A'] # 多索引查询
df.query('i0 == "b" & i1 == "b"') # 多索引查询方法 2
# 取多索引中指定级别的所有不重复值
df.index.get_level_values(2).unique()
# 去掉为零小数，12.00 -> 12
df.astype('str').applymap(lambda x: x.replace('.00', ''))
# 插入数据，在第三列加入「两倍」列
df.insert(3, '两倍', df['值']*2)
# 枚举转换
df['gender'] = df.gender.map({'male':'男', 'female':'女'})
# 增加本行之和列
df['Col_sum'] = df.apply(lambda x: x.sum(), axis=1)
# 对指定行进行加和
col_list= list(df)[2:] # 取请假范围日期
df['总天数'] = df[col_list].sum(axis=1) # 计算总请假天数
# 对列求和，汇总
df.loc['col_sum'] = df.apply(lambda x: x.sum())
# 按指定的列表顺序显示
df.reindex(order_list)
# 按指定的多列排序
df.reindex(['col_1', 'col_5'], axis="columns")
```

07.数据选取

```
df[col] # 根据列名，并以Series的形式返回列
df[[col1, col2]] # 以DataFrame形式返回多列
df.loc[df['team'] == 'B',['name']] # 按条件查询，只显示name 列
s.iloc[0] # 按位置选取数据
s.loc['index_one'] # 按索引选取数据
df.loc[0,'A':'B'] #  A到 B 字段的第一行
df.loc[2018:1990, '第一产业增加值':'第三产业增加值']
df.loc[0,['A','B']] # d.loc[位置切片，字段]
df.iloc[0,:] # 返回第一行，iloc 只能是数字
df.iloc[0,0] # 返回第一列的第一个元素
dc.query('site_id > 8 and utype=="老客"').head() # 可以 and or / & |
 # 迭代器及使用
for idx,row in df.iterrows(): row['id']
# 迭代器对每个元素进行处理
df.loc[i,'链接'] = f'http://www.gairuo.com/p/{slug}.html'
for i in df.Name:print(i) # 迭代一个列
# 按列迭代，[列名，列中的数据序列 S（索引名 值)]
for label, content in df.items():print(label, content)
# 按行迭代，迭代出整行包括索引的类似列表的内容，可row[2]取
for row in df.itertuples():print(row)

df.at[2018, '总人口'] # 按行列索引名取一个指定的单个元素
df.iat[1, 2] # 索引和列的编号取单个元素
s.nlargest(5).nsmallest(2) # 最大和最小的前几个值
```

```
df.nlargest(3, ['population', 'GDP'])
df.take([0, 3]) # 指定多个行列位置的内容
# 按行列截取掉部分内容，支持日期索引标签
ds.truncate(before=2, after=4)
# 将 dataframe 转成 series
df.iloc[:,0]
float(str(val).rstrip('%')) # 百分数转数字
df.reset_index(inplace=True) # 取消索引
```

08.数据处理 GroupBy 透视

```
df.groupby(col) # 返回一个按列col进行分组的Groupby对象
df.groupby([col1,col2]) # 返回一个按多列进行分组的Groupby对象
df.groupby(col1)[col2] # 返回按列col1进行分组后，列col2的均值
# 创建一个按列col1进行分组，并计算col2和col3的最大值的数据透视表
df.pivot_table(index=col1,
               values=[col2,col3],
               aggfunc=max,
               as_index=False)
# 同上
df.pivot_table(index=['site_id', 'utype'],
               values=['uv_all', 'regist_num'],
               aggfunc=['max', 'mean'])
df.groupby(col1).agg(np.mean) # 返回按列col1分组的所有列的均值
# 按列将其他列转行
pd.melt(df, id_vars=["day"], var_name='city', value_name='temperature')
# 交叉表是用于统计分组频率的特殊透视表
pd.crosstab(df.Nationality,df.Handedness)
# groupby 后排序，分组 agg 内的元素取固定个数
(
    df[(df.p_day >= '20190101')]
    .groupby(['p_day', 'name'])
    .agg({'uv':sum})
    .sort_values(['p_day','uv'], ascending=[False, False])
    .groupby(level=0).head(5) # 每天取5个页面
    .unstack()
    .plot()
)
# 合并查询经第一个看（max，min，last，size:数量）
df.groupby('结算类型').first()
# 合并明细并分组统计加总（'max'，`mean`，`median`，
# `prod`，`sum`，`std`,`var`，'nunique'），'nunique'为去重的列表
df1 = df.groupby(by='设计师ID').agg({'结算金额':sum})
df.groupby(by=df.pf).ip.nunique() # groupby distinct，分组+去重数
df.groupby(by=df.pf).ip.value_counts() # groupby 分组+去重的值及数量
df.groupby('name').agg(['sum', 'median', 'count'])
```

09.数据合并

```
# 合并拼接行
# 将df2中的行添加到df1的尾部
df1.append(df2)
# 指定列合并成一个新表新列
ndf = (df['提名1']
       .append(df['提名2'], ignore_index=True)
       .append(df['提名3'], ignore_index=True))
ndf = pd.DataFrame(ndf, columns=(['姓名']))
# 将df2中的列添加到df1的尾部
df.concat([df1, df2], axis=1)

# 合并文件的各行
df1 = pd.read_csv('111.csv', sep='\t')
df2 = pd.read_csv('222.csv', sep='\t')
excel_list = [df1, df2]
# result = pd.concat(excel_list).fillna('')[:].astype('str')
result = pd.concat(excel_list)[]
result.to_excel('333.xlsx', index=False)

# 合并指定目录下所有的 excel (csv) 文件
import glob
files = glob.glob("data/cs/*.xls")
dflist = []
for i in files:
    dflist.append(pd.read_excel(i, usecols=['ID', '时间', '名称']))

df = pd.concat(dflist)

# 合并增加列
# 对df1的列和df2的列执行SQL形式的join
df1.join(df2,on=col1,how='inner')
# 用 key 合并两个表
df_all = pd.merge(df_sku, df_spu,
                  how='left',
                  left_on=df_sku['product_id'],
                  right_on=df_spu['p.product_id'])
```

10.时间处理 时间序列

```
# 时间索引
df.index = pd.DatetimeIndex(df.index)

# 时间只保留日期
df['date'] = df['time'].dt.date
```

```python
# 将指定字段格式化为时间类型
df["date"] = pd.to_datetime(df['时间'])

# 转化为北京时间
df['time'] = df['time'].dt.tz_convert('Asia/Shanghai')

# 转为指定格式，可能会失去秒以后的精度
df['time'] = df['time'].dt.strftime("%Y-%m-%d %H:%M:%S")
dc.index = pd.to_datetime(dc.index, format='%Y%m%d', errors='ignore')

# 时间，参与运算
pd.DateOffset(days=2)

# 当前时间
pd.Timestamp.now()
pd.to_datetime('today')

# 判断时间是否当天
pd.datetime.today().year == df.start_work.dt.year
df.time.astype('datetime64[ns]').dt.date == pd.to_datetime('today')

# 定义个天数
import datetime
days = lambda x: datetime.timedelta(days=x)
days(2)

# 同上，直接用 pd 包装的
pd.Timedelta(days=2)

# unix 时间戳
pd.to_datetime(ted.film_date, unit='ms')

# 按月（YMDHminS）采集合计数据
df.set_index('date').resample('M')['quantity'].sum()
df.set_index('date').groupby('name')['ext price'].resample("M").sum()

# 按天汇总，index 是 datetime 时间类型
df.groupby(by=df.index.date).agg({'uu':'count'})

# 按周汇总
df.groupby(by=df.index.weekday).uu.count()

# 按月进行汇总
df.groupby(['name', pd.Grouper(key='date', freq='M')])['ext price'].sum()

# 按月进行汇总
df.groupby(pd.Grouper(key='day', freq='1M')).sum()

# 按照年度，且截止到12月最后一天统计 ext price 的 sum 值
df.groupby(['name', pd.Grouper(key='date', freq='A-DEC')])['ext price'].sum()
```

```
# 按月的平均重新采样
df['Close'].resample('M').mean()

# 取时间范围，并取工作日
rng = pd.date_range(start="6/1/2016",end="6/30/2016",freq='B')

# 重新定时数据频度，按一定补充方法
df.asfreq('D', method='pad')

# 时区，df.tz_convert('Europe/Berlin')
df.time.tz_localize(tz='Asia/Shanghai')

# 转北京时间
df['Time'] = df['Time'].dt.tz_localize('UTC').dt.tz_convert('Asia/Shanghai')

# 查看所有时区
from pytz import all_timezones
print (all_timezones)

# 时长，多久，两个时间间隔时间，时差
df['duration'] = pd.to_datetime(df['end']) - pd.to_datetime(df['begin'])

# 指定时间进行对比
df.Time.astype('datetime64[ns]') < pd.to_datetime('2019-12-11 20:00:00', format='%Y-
%m-%d %H:%M:%S')
```

11.常用备忘

```
# 解决科学计数法问题
df = pd.read_csv('111.csv', sep='\t').fillna('')[:].astype('str')
# 和订单量相关性最大到小显示
dd.corr().total_order_num.sort_values(ascending=False)

# 解析列表、json 字符串
import ast
ast.literal_eval("[{'id': 7, 'name': 'Funny'}]")

# Series apply method applies a function to
# every element in a Series and returns a Series
ted.ratings.apply(str_to_list).head()
# lambda is a shorter alternative
ted.ratings.apply(lambda x: ast.literal_eval(x))
# an even shorter alternative is to apply the
# function directly (without lambda)
ted.ratings.apply(ast.literal_eval)
# 索引 index 使用 apply()
df.index.to_series().apply()
```

12.样式显示

```
df['per_cost'] = df['per_cost'].map('{:,.2f}%'.format)  # 显示%比形式
# 指定列表（值大于0）加背景色
df.style.applymap(lambda x: 'background-color: grey' if x>0 else '',
                  subset=pd.IndexSlice[:, ['B', 'C']])

# 最大值最小值加背景色
df.style.highlight_max(color='lightgreen').highlight_min(color='#cd4f39')
df.style.format('{:.2%}', subset=pd.IndexSlice[:, ['B']]) # 显示百分号

# 指定各列的样式
format_dict = {'sum':'${0:,.0f}',
                       'date': '{:%Y-%m}',
                       'pct_of_total': '{:.2%}'
                       'c': str.upper}

# 一次性样式设置
(df.style.format(format_dict) # 多种样式形式
    .hide_index()
    # 指定列按颜色深度表示值大小，cmap 为 matplotlib colormap
    .background_gradient(subset=['sum_num'], cmap='BuGn')
    # 表格内作横向 bar 代表值大小
    .bar(color='#FFA07A', vmin=100_000, subset=['sum'], align='zero')
    # 表格内作横向 bar 代表值大小
    .bar(color='lightgreen', vmin=0, subset=['pct_of_total'], align='zero')
    # 下降（小于0）为红色，上升为绿色
    .bar(color=['#ffe4e4','#bbf9ce'], vmin=0, vmax=1, subset=['增长率'],
align='zero')
    # 给样式表格起个名字
    .set_caption('2018 Sales Performance')
    .hide_index())

# 按条件给整行加背景色（样式）
def background_color(row):
    if row.pv_num >= 10000:
        return ['background-color: red'] * len(row)
    elif row.pv_num >= 100:
        return ['background-color: yellow'] * len(row)
    return [''] * len(row)
# 使用
df.style.apply(background_color, axis=1)
```

13.表格中的直方图，sparkline 图形

```
# 表格中的直方图,sparkline 图形
import sparklines
```

```
import numpy as np
def sparkline_str(x):
    bins=np.histogram(x)[0]
    sl = ''.join(sparklines.sparklines(bins))
    return sl
sparkline_str.__name__ = "sparkline"
# 画出趋势图，保留两位小数
df.groupby('name')['quantity', 'ext price'].agg(['mean', sparkline_str]).round(2)

# sparkline 图形

def sparkline(data, figsize=(4, 0.25), **kwargs):
    """
    creates a sparkline
    """

    # Turn off the max column width so the images won't be truncated
    pd.set_option('display.max_colwidth', -1)

    # Turning off the max column will display all the data
    # if gathering into sets / array we might want to restrict to a few items
    pd.set_option('display.max_seq_items', 3)

    #Monkey patch the dataframe so the sparklines are displayed
    pd.DataFrame._repr_html_ = lambda self: self.to_html(escape=False)

    from matplotlib import pyplot as plt
    import base64
    from io import BytesIO

    data = list(data)

    *_, ax = plt.subplots(1, 1, figsize=figsize, **kwargs)
    ax.plot(data)
    ax.fill_between(range(len(data)), data, len(data)*[min(data)], alpha=0.1)
    ax.set_axis_off()

    img = BytesIO()
    plt.savefig(img)
    plt.close()
    return '<img src=" {}"
/>'.format(base64.b64encode(img.getvalue()).decode())

# 使用
df.groupby('name')['quantity', 'ext price'].agg(['mean', sparkline])
df.apply(sparkline, axis=1) # 仅支持横向数据画线，可做 T 转置
```

14.可视化

```
kind : str
- 'line' : line plot (default)
- 'bar' : vertical bar plot
- 'barh' : horizontal bar plot
- 'hist' : histogram
- 'box' : boxplot
- 'kde' : Kernel Density Estimation plot
- 'density' : same as 'kde'
- 'area' : area plot
- 'pie' : pie plot
```

15.常用方法

```
df88.plot.bar(y='rate', figsize=(20, 10)) # 图形大小，单位英寸
df_1[df_1.p_day > '2019-06-01'].plot.bar(x='p_day', y=
['total_order_num','order_user'], figsize=(16, 6)) # 柱状图
# 每条线一个站点，各站点的 home_remain，stack的意思是堆叠，堆积
# unstack 即“不要堆叠”
(df[(df.p_day >= '2019-05-1') & (df.utype == '老客')].groupby(['p_day', 'site_id'])
['home_remain'].sum().unstack().plot.line())
#  折线图，多条，x 轴默认为 index
dd.plot.line(x='p_day', y=['uv_all', 'home_remain'])
dd.loc['新访客', 2].plot.scatter(x='order_user', y='paid_order_user') # 散点图
dd.plot.bar(color='blue') # 柱状图，barh 为横向柱状图
sns.heatmap(dd.corr()) # 相关性可视化
#  刻度从0开始，指定范围 ylim=(0,100)，x 轴相同
s.plot.line(ylim=0)

# 折线颜色 https://matplotlib.org/examples/color/named_colors.html
# 样式( '-','--','-.',':' )
# 折线标记 https://matplotlib.org/api/markers_api.html
# grid=True 显示刻度 etc:
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html
s.plot.line(color='green', linestyle='-', marker='o')

# 两个图绘在一起
[df['数量'].plot.kde(), df['数量'].plot.hist()]

# 对表中的数据按颜色可视化
import seaborn as sns
cm = sns.light_palette("green", as_cmap=True)
df.style.background_gradient(cmap=cm, axis=1)

# 将数据转化为二维数组
[i for i in zip([i.strftime('%Y-%m-%d') for i in s.index.to_list()], s.to_list())]

# 和 plot 用法一样 https://hvplot.pyviz.org/user_guide/Plotting.html
```

```
import hvplot.pandas

# 打印 Sqlite 建表语句
print(pd.io.sql.get_schema(fdf, 'table_name'))
```

16.Jupyter notebooks 问题

```
# jupyter notebooks plt 图表配置
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (15.0, 8.0) # 固定显示大小
plt.rcParams['font.family'] = ['sans-serif'] # 显示中文问题
plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文问题

# jupyter notebooks 页面自适应宽度
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
# 背景白色 <style>#notebook_panel {background: #ffffff;}</style>

# jupyter notebooks 嵌入页面内容
from IPython.display import IFrame
IFrame('https://arxiv.org/pdf/1406.2661.pdf', width=800, height=450)

# Markdown 一个 cell 不支持多张粘贴图片
# 一个文件打印打开只显示一张图片问题解决
# /site-packages/notebook/static/notebook/js/main.min.js var key 处
# 33502、33504 行
key = utils.uuid().slice(2,6)+encodeURIandParens(blob.name);
key = utils.uuid().slice(2,6)+Object.keys(that.attachments).length;
#
https://github.com/ihnorton/notebook/commit/55687c2dc08817da587977cb6f19f8cc0103bab1

# 多行输出
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all' #默认为'last'

# 执行 shell 命令: ! <命令语句>

# 在线可视化工具
https://plot.ly/create
```

17.Slideshow 幻灯片

```
安装 RISE 库: pip install RISE
- [Alt+r] 播放/退出播放
-「,」逗号隐藏左侧两个大操作按钮，「t」总览 ppt，「/」黑屏
- Slide：主页面，通过按左右方向键进行切换。
```

- Sub-Slide：副页面，通过按上下方向键进行切换。全屏
- Fragment：一开始是隐藏的，按空格键或方向键后显示，实现动态效果。在一个页面
- Skip：在幻灯片中不显示的单元。
- Notes：作为演讲者的备忘笔记，也不在幻灯片中显示

Col1 Col2 Col3