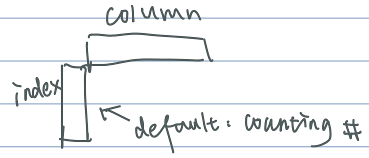
Topic1: Data Structure and Methods of Series



! pip install plotly | import numpy as np / scipy as sp / pandas as pd

Data structure Series and DataFrame (DF contains multiple columns of Series)

- sales[0] / sales.loc["Jan"] ← Jan's value. - logic selection. m = sales.median(), mask = sales > m, sales[mask] 输出>m的行
- NaN / np.nan. - df.SoldFlag / df["SoldFlag"] (输出Series) - 按Ctrl + Space会出现可选择的方法
- Methods : flag.max() - Attributes : flag.index / .real (real part of a number)

① Operators ② Aggregate methods (reduce to a value): .mean(), .max(), .sum(), .is_monotonic() ③ Conversion methods: .to_datetime() (export data to other format)

④ Manipulation methods: .sort_values, .drop_duplicates (return Series with same index) ⑤ Indexing and accessor methods: .loc, .iloc. (return Series or scalars)

⑥ Ploting: .plot ⑦ Transformation method: .unstack, .reset_index, .agg, .transform, .pivot ⑧ Attribute: .index, .dtype.

Algebraic Operation: column-wisely +, -, *, /, % 取余modulus, @ matrix multiplication, ** power, <, >, &, |. -gt(0)大于0的

! list x2 就会成 [1,2] → [1,2,1,2], 而Series x2是 1.2 → 2.4 - 直接用 A+B, 只要一边没有值就会加和成 NaN, 但 A.add(B, fill_value=0) (把没有的都当成0 8+0) .div(100)

(1 nan / 2 交 / 2 叉 / 1 2 2 加 / 2 + 2 ⇒ 2 / 2 4 4 nan)

Aggregation and Conversion: - city.quantile([.1, .8, .99]) -argmax() (第一个最大的位置), count(), skew(), std(), var() (ddof=0(pop)/1(sam)), -.agg(["mean"]) (单个不用) .is_unique() ← 单增

-value_count() → count 每个数值的个数. 找分布. - A.astype("float32") 换数据类型 - history.index = pd.to_datetime(history.index) 换成时间格式

Manipulation - A.gt(50) / A.apply(def) 一个效果. 但apply慢 - A.where(A.le(50), other=0) 保留<50的, 否则为0 - Missing: .isna() / .isnull() / .dropna()

-.ffill() 用昨天的数据, bfill() 用明天的, .interpolate() 前后平均 - clip(lower=, upper=) 所有极值都用给定数字替换. -.sort_values(ascending=False) 大到小 / sort_index()

- drop_duplicates(keep=) (subset=" ") 本身留第一个 {"last" 留最后一个 / False 全部扔掉} - rank() 不动顺序下, 对值排序 (0 40, 1 20, 2 30 → 3, 1, 2) (重复: () → 1, 2.5, 2.5, 4; method='min' → 1, 2, 2, 4; 'dense' → 1, 2, 2, 3)

- A.replace([1,2], "lowest"), (to_replace=[4,1], value=[3,2]) / (to_replace={4:3, 1:2}) / (to_replace='z.*', value='zanne', regex=True): Suzy → Suzanne

- Binning: .cut(A, bins=4) / (A, [-1,0,1,2,3]) 返回每个所在区间: 0 (1,2] 1 (2,3]

Indexing Operation -s.rename(index={0:'first'}), A.reset_index(drop=True) (return a series) 先加了一列从0起, 再把原本的删了. 但 .reset_index() (原本的成了一列"index" return dataframe)

:rename_axis("column") -.loc["A"] scalar / .loc[["A"]] series. - s.sort_index().loc["G":"P"] 把从G到P的列取出来 (但若要pxx, 需写更远的终止")

- cost.mul(1.1).loc[gt3] (写函数. 作用在mul后) / .loc[cost>3] (作用在 ori cost上) - A.iloc[0] 值, .iloc[[0]] 一行, .iloc[[1,-2]], 第二行, 倒数第二行, [0:3] / [-2:] (←尾不包含) -.head(10), .tail(10)

- A.sample(10) 抽10个出来, .sample(frac=0.5, replace=True) - A.filter(like="K") 把"K"相关的行列出来, case sensitive. - A.reindex(["A", "B"]) (A 3 / B NaN)

Date and Time - date.dt.day_name() Sunday / Friday, dt.is_month_end() True / False, dt.strftime('%m/%d/%y') (01/01/13), .loc[:"01/01/13"]

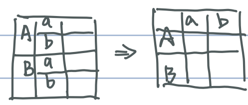
- Shifting temp.shift(1) 第一行变成 NaN - Rolling data.rolling(3).mean() 表示本身+之前2个. -.cumsum() 累和和. cummax()

-.expanding() 从头开始加和, 和rolling()一样可选min_periods 默认为1. - resemble("Q"/"2M") 季度/每2个月 (用于时间)

t.sum() / .agg("sum") 就是2M 输出一个和, t.transform("sum") 就把这2M的60天都转成一个值.

Plotting - np.random.normal(10, 5, 5000) 画出5000个 u=10, 6=5 - ① matplotlib: plt.plot 是 line, .hist

- ② pandas: temp.plot.hist(), .box(), .kde(), .line(), -. unstack() 会将最内层的 变成列



用pandas的好处是可以进行前序的操作, 如 sector.value_count().plot.barh()

s.agg(func=None, axis=0, *args, **kwargs)	Returns a scalar if func is a single aggregation function. Returns a series if a list of aggregations are passed to func.
s.all(axis=0, bool_only=None, skipna=True, level=None)	Returns True if every value is truthy. Otherwise False
s.any(axis=0, bool_only=None, skipna=True, level=None)	Returns True if at least one value is truthy. Otherwise False
s.autocorr(lag=1)	Returns Pearson correlation between s and shifted s
s.corr(other, method='pearson')	Returns correlation coefficient for 'pearson', 'spearman', 'kendall', or a callable.
s.cov(other, min_periods=None)	Returns covariance.
s.max(axis=None, skipna=None, level=None, numeric_only=None	Returns maximum value.
s.min(axis=None, skipna=None, level=None, numeric_only=None)	Returns minimum value.
s.mean(axis=None, skipna=None, level=None, numeric_only=None)	Returns mean value.
s.median(axis=None, skipna=None, level=None, numeric_only=None)	Returns median value.
s.prod(axis=None, skipna=None, level=None, numeric_only=None, min_count=0)	Returns product of s values.

s.quantile(q=.5, interpolation='linear')	Returns 50% quantile by default. *Note* returns Series if q is a list.
s.sem(axis=None, skipna=None, level=None, ddof=1, numeric_only=None)	Returns unbiased standard error of mean.
s.std(axis=None, skipna=None, level=None, ddof=1, numeric_only=None)	Returns sample standard deviation.
s.var(axis=None, skipna=None, level=None, ddof=1, numeric_only=None)	Returns unbiased variance.
s.skew(axis=None, skipna=None, level=None, numeric_only=None)	Returns unbiased skew.
s.kurtosis(axis=None, skipna=None, level=None, numeric_only=None)	Returns unbiased kurtosis.
s.nunique(dropna=True)	Returns count of unique items.
s.count(level=None)	Returns count of non-missing items.
s.size	Number of items in series. (Property)
s.is_unique	True if all values are unique
s.is_monotonic	True if all values are increasing
s.is_monotonic_increasing	True if all values are increasing
s.is_monotonic_decreasing	True if all values are decreasing

r.apply(func, args=None, kwargs=None)	Apply custom aggregation function to rolling group.
r.corr(other, method='pearson')	Returns correlation coefficient for 'pearson', 'spearman', 'kendall', or a callable.
r.count(other, method='pearson')	Returns count of non NaN values.
r.cov(other, min_periods=None)	Returns covariance.
r.max(axis=None, skipna=None, level=None, numeric_only=None	Returns maximum value.
r.min(axis=None, skipna=None, level=None, numeric_only=None)	Returns minimum value.
r.mean(axis=None, skipna=None, level=None, numeric_only=None)	Returns mean value.
r.median(axis=None, skipna=None, level=None, numeric_only=None)	Returns median value.
r.quantile(q=.5, interpolation='linear')	Returns 50% quantile by default. *Note* returns Series if q is a list.
r.sem(axis=None, skipna=None, level=None, ddof=1, numeric_only=None)	Returns unbiased standard error of mean.
r.std(axis=None, skipna=None, level=None, ddof=1, numeric_only=None)	Returns sample standard deviation.
r.var(axis=None, skipna=None, level=None, ddof=1, numeric_only=None)	Returns unbiased variance.
r.skew(axis=None, skipna=None, level=None, numeric_only=None)	Returns unbiased skew.

.date	Property with a series of Python datetime.date objects.
.day	Property with a series of day of month.
.day_name(locale='en_us')	Return the string day of week.
.dayofweek	Property with a series of date of week as number (0 is Monday).
.dayofyear	Property with a series of day of the year.
.days_in_month	Property with a series of number of days in month.
.daysinmonth	Property with a series of number of days in month.
.floor(freq=None, ambiguous=None, nonexistent=None)	Return floor according to offset alias in freq. The nonexistent parameter controls DST time issues.
.hour	Property with a series of hour of date.
.is_leap_year	Property with a series of booleans if date is leap year.
.is_month_end	Property with a series of booleans if date is end of month.
.is_month_start	Property with a series of booleans if date is start of month.
.is_quarter_end	Property with a series of booleans if date is end of quarter.
.is_quarter_start()	Property with a series of booleans if date is start of quarter.
.is_year_end	Property with a series of booleans if date is end of year.

Topic2: Working With DataFrames

DataFrame – df["xx"]是series的格式 –.index(), .shape(), .columns(), .describe(), .axes(): 0是index, 1是column

df.sum(axis=0)/(axis="index"), df.sort_index(by="x", ascending=False), df.set_index("x"), df['A'].value_count() → normalize

df["Season"]=[1 if t<4 else 2 if t<7 else 3 if t<10 else 4 for t in df["Month"]] 填有n个M.nN

df.rename(columns=lambda n:n+"0").head()/.assign(New=lambda x: x.AEP*100), .replace([1,2],np.nan).head()

df1+df2 其中一个没有就是NaN, .concat([df1,df2],axis=1) 0就是上下拼接, 1是左右, .append().append().. .Pct_change()

Aggregation: df.agg(["count","mean"],axis=0], df.apply(np.sum, axis=0)

Missing & duplicated values: .isna(), .isnull().sum()每行.sum()总的. df.loc[df.index[3]]=3, .dropna()默认为行

Filtering & subset selection – df[["A","B"]], loc是左包含, iloc右是不包含的, df[df["AEP"]>15000]

df.query("AEP>15000 and COMED>12000"), mask=df["AEP"]>15000, df.query("@mask and COMED>12000")

df.loc[df.AEP>24000, df.columns[-3:]]

Groupby Operation – df.groupby("WeekDay")["AEP"].mean()返回七个时间的AEP平均值.

df.groupby("WeekDay")["AEP"].transform(lambda x: (x-np.mean(x))/np.std(x)) 对每个值,减去相对应group的 mean std.

df.groupby("A").filter(lambda x: x["B"].mean()>3): 把df分成M,N组,选其中B列的mean>3的组(M的所有行)

Pivoting:

Year New
1990 0 10
1 20
1994 0 30
1 40

sf.pivot_table(index="Year", columns="New", values="Price", aggfunc="mean")

pd.crosstab(index=sf.Year, columns=sf.New, values=sf.Price, aggfunc="mean")

sf.groupby(["Year","New"]).Price.mean().unstack()

New	0	1
Year		
1990	10	20
1994	30	40

Data Assembly ⇒ performance.melt(id_vars=["Name","ID"],

value_vars=["Python","Statistics"], var_name="Course", value_name="Score")

⇐ pmelt.pivot_table(index=["Name","ID"], columns="Course", values="Score")

Name	ID	Py	Sta
A	1	60	80
B	2	23	82

⇔

Name	ID	Course	Score
A	1	Py	60
B	2	Py	23
A	1	Sta	80
B	2	Sta	82

– transposing data: df.T

Interactive Plotting px.histgram(), .scatter(), .line()

– import plotly.graph_objects as go. lay=go.Layout(), fig=go.Figure() fig.show()

fig.add_trace(), fig.update_layout() fig=make_subsets()

.is_year_start	Property with a series of booleans if date is start of year.
.microsecond	Property with a series of microseconds of date.
.minute	Property with a series of minutes of date.
.month	Property with a series of month of date (numeric).
.month_name(locale='en_us')	Return a series of month of date (string).
.nanosecond	Property with a series of nanoseconds of date.
.normalize()	Return a series of dates converted to midnight.
.quarter	Property with series of quarter of date (numeric 1-4).
.round(freq=None, ambiguous=None, nonexistent=None)	Return round according to fixed frequency (cannot be end like 'ME') in freq. The nonexistent parameter controls DST time issues.
.second	Property with a series of seconds of date (numeric).
.strftime(date_format)	Return a series with string dates. Formatted using strftime format codes.
.time	Property with a series of Python datetime.time objects.
.timetz	Property with a series of Python datetime.time objects with timezone information.
.to_period(freq)	Return a series with pandas Period objects.
.to_pydatetime()	Return a numpy array with datetime.datetime objects.
.tz	Property with timezone.
.tz_convert(tz)	Convert from one timezone aware series to another.
.tz_localize(tz, ambiguous=None, nonexistent=None)	Convert from naive to timezone aware.

s.plot(ax=None, style=None, logx=False, logy=False, xticks=None, yticks=None, xlim=None, ylim=None, xlabel=None, ylabel=None, rot=None, fontsize=None, colormap=None, table=False, **kwargs)	Common plot parameters. Use ax to use existing Matplotlib axes, style for color and marker style (see matplotlib.marker), _ticks to specify tick locations, _lim to specify tick limits, _label to specify x/y label (default to index/column name), rot to rotate labels, fontsize for tick label size, colormap for coloring, position, table to create table with data. Additional arguments are passed to plt.plot
s.plot.bar(position=.5, color=None)	Create a bar plot. Use position to specify label alignment (0-left, 1-right). Use color (string, list) to specify line color.
s.plot.barh(x=None, y=None, color=None)	Create a horizontal bar plot. Use position to specify label alignment (0-left, 1-right). Use color (string, list) to specify line color.
s.plot.hist(bins=10)	Create a histogram. Use bins to change the number of bins.
s.plot.box()	Create a boxplot.
s.plot.kde(bw_method='scott', ind=None)	Create a Kernel Density Estimate plot. Use bw_method to calculate estimator bandwidth (see scipy.stats.gaussian_kde). Use ind to specify evaluation points for PDF estimation (NumPy array of points, or integer with equally spaced points).
s.plot.line(color=None)	Create a line plot. Use color to specify line color.
s.plot.pie()	Create a pie plot.

Operator methods instead of operators

Pandas also provides methods for the standard operators.

Method	Operator	Description
s.add(s2)	s + s2	Adds series
s.radd(s2)	s2 + s	Adds series
s.sub(s2)	s - s2	Subtracts series
s.rsub(s2)	s2 - s	Subtracts series
s.mul(s2) s.multiply(s2)	s * s2	Multiplies series
s.rmul(s2)	s2 * s	Multiplies series
s.div(s2) s.truediv(s2)	s / s2	Divides series

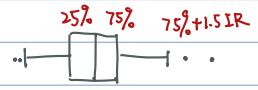
Ignore
radd,rsub,rmul,truediv,rmod,rfloor
,rpow,

s.mod(s2)	s % s2	Modulo of series division
s.rmod(s2)	s2 % s	Modulo of series division
s.floordiv(s2)	s // s2	Floor divides series
s.rfloordiv(s2)	s2 // s	Floor divides series
s.pow(s2)	s ** s2	Exponential power of series
s.rpow(s2)	s2 ** s	Exponential power of series
s.eq(s2)	s2 == s	Elementwise equals of series
s.ne(s2)	s2 != s	Elementwise not equals of series
s.gt(s2)	s > 2	Elementwise greater than of series
s.ge(s2)	s >= 2	Elementwise greater than or equals of series
s.lt(s2)	s < 2	Elementwise less than of series
s.le(s2)	s <= 2	Elementwise less than or equals of series
np.invert(s)	~s	Elementwise inversion of boolean series (no pandas method).
np.logical_and(s, s2)	s & s2	Elementwise logical and of boolean series (no pandas method).
np.logical_or(s, s2)	s | s2	Elementwise logical or of boolean series (no pandas method).

'all'	Returns True if every value is truthy.
'any'	Returns True if any value is truthy.
'autocorr'	Returns Pearson correlation of series with shifted self. Can override lag as keyword argument(default is 1).
'corr'	Returns Pearson correlation of series with other series. Need to specify other.
'count'	Returns count of non-missing values.
'cov'	Return covariance of series with other series. Need to specify other.
'dtype'	Type of the series.
'dtypes'	Type of the series.
'empty'	True if no values in series.
'hasnans'	True if missing values in series.

Topic3: Reshaping Statistics with Python

Population and Sample - data['A'].sample(10, replace=True)



μ: mean(), σ^2: var(ddof=0), σ: std(ddof=0), size: shape[0]

Variable and Distribution Numerical, Categorical. IR: A.quantile(0.75) - A.quantile(0.25)

Kernel Density Estimation: $\hat{f}(x;h) = \frac{1}{nh}\sum K(\frac{x-x_i}{h})$ kernel function: $\int K(x)dx = 1$ Gaussian, Rectangular, Triangular, uniform

h: bandwidth small h: small bias, large var (noise), model.score_sample(): output the log function of the densities.

from sklearn.neighbors import KernelDensity KernelDensity(kernel='gaussian', bandwidth=0.1),

- 对于厚度(fat tail)的 stats.t.fit(data)[0] 输出 df=3.28. (t越大越接近正态)

Confidence interval and Hypothesis Testing: $\bar{x} \to \mu$, $\frac{\bar{x}-\mu}{s/\sqrt{n}} \to \sigma$ $[\bar{x} - z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}, \bar{x} + z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}]$ $p<0.05$ 拒绝正态.

normality: histogram to visual, stats.probplot(data, dist="norm") 是Q-Qplot, S-W test: shapiro(data): H_0: normal.

Kolmogorov-Smirnov Test: Ktest(data, stats.norm.cdf) H_0: from the same distribution

xbar=.mean(), S=.std(ddof=1) n=.shape[0] zhat=(Xbar-0)/(S/(n**0.5)), p-value=1-norm.cdf(zhat, 0.1)

Association: .corr(), scatter_matrix(), corr() 包里有 "pearson", "spearman", "kendall" (Similar)

Pearson correlation: can find linear pattern, Spearman's Rank-Order correlation: association based on rank [-1,1]

Hoeffding Dependence Coefficient: sign is not interpretation, identify non-mono [-0.5,1], Distance Coefficient: 0 is indep [0,1]

Extreme Value Theory: Kurtosis $E(\frac{x-\mu}{\sigma})^4$

Fisher-Tippett-Gnedenko Theorem iid. $M_n = \max\{X_1,\dots,X_n\} \approx GEV(\mu,\sigma,\xi)$ (General Extreme Value Dist)

$F(s) = \begin{cases} \exp(-\exp(-s)) & \xi=0 \\ \exp(-(1+\xi s)^{-\frac{1}{\xi}}) & \xi\neq 0 \end{cases}$ $s = \frac{x-\mu}{\sigma}$

gev.pdf(x, c=$-\xi$, loc=0, scale=1)

- μ is location parameter. shift left/right
- σ is scale parameter. the spread of the distribution
- ξ is extreme value index (EVI), shape parameter, determine shape of distribution

$x < \mu - \frac{\sigma}{\xi}$. short tailed. $\xi<0$. Weibull Dist. nega.infi

$\xi>0$. Fr'echet Dist. positive infinite $(x > \mu - \frac{\sigma}{\xi})$ Heavy tailed; $\xi=0$, Gumbel Dist. posi&nega infinite. light tailed.

How to sample? -Block Maxima (BM) 每一段时间的最大(不会有关联性)选区间) -Point Above Threshold (POT) (取所有大值)

c, loc, scale = gev.fit(data, loc=0.2, scale=0.2) set initial guess

Extreme Score: tail prob of GEV. 有多少概率会更极端 (损失超-0.05) 1-gev.cdf(data, c, loc, scale)

Rare event: extreme score is small 明明预测下雨的概率0.1%为0, 可还是下了

Extreme Value at Risk: 如果算出 95% EVaR = -0.06069, 则只有5%的概率损失会大于6.069%

Topic4: Predictive Model

- $SST = \sum(Y_i - \bar{Y})^2$, $SSR = \sum(\hat{Y}_i - \bar{Y})^2$ $SSE = \sum(Y_i - \hat{Y}_i)^2$ $R^2_{adj} = [R^2 - \frac{k}{n-1}][\frac{n-1}{n-k-1}] = 1 - \frac{SSE/(n-k-1)}{SST/(n-1)}$

$MSE = \frac{e_1^2 + \dots + e_n^2}{n-k-1}$ - overfitting / underfitting. 看 train/test 的 R^2_{adj}, RMSE, confusion matrix 的比例

- Nonlinear: ① X_1^2, X_2^2, X_1X_2. ② $\frac{X_1}{X_2}, \frac{X_2}{X_3}$

Import dcor. dcor.distance_correlation $F(model) = \frac{SSR/k}{SSE/[n-(k+1)]}$ $F > 2.5$ then reject $H_0: \beta_1 \dots \beta_k = 0$

model.bse: sd of the parameter. model["pvalue"] = 2 * norm.cdf(-np.abs(model.params/model.bse))

$P(Y=1|X) = \frac{e^{x\theta}}{1+e^{x\theta}}$ $\log(\frac{p}{1-p}) = \beta_0 + \beta_1 X$

$$PPV = P(Real\ 1|Predict\ 1)$$
$$NPV = P(Real\ 0|Predict\ 0)$$
$$Sensitivity = P(Predict\ 1|Real\ 1) = TPR, \text{ True Positive Rate, } \mathbf{TPR}$$
$$Specificity = P(Predict\ 0|Real\ 0)$$
$$1 - Specificity = P(Predict1|Real\ 0) = FPR, \text{ False Positive Rate, } \mathbf{FPR}$$

	realY=0	realY=1	Measure
predictedY=0	True Negative TN	False Negative FN	Negative predictive value(NPV) $\frac{TN}{FN+TN}$
predictedY=1	Flase Positive FP	True Positive TP	Positive predictive value(NPV) $\frac{TP}{FP+TP}$
Measure	specificity $\frac{TN}{FP+TN}$	Sensitivity $\frac{TP}{TP+FN}$	Accuracy $\frac{TP+TN}{TP+FP+FN+TN}$