تمرین سری یک:

ائلدار صمدزاده طریقت ٤٠٠٣٣٠٤٤

۱- مثال های فصل ۱ کتاب مرجع اصلی را در یک فایل نوت بوک پیاده سازی کنید و توضیحات کافی را به آن اضافه کنید.

۲- فایل نوت بوک exercise 1 که ضمیمه این تمرین هست را اجرا کنید، با ضبط یک یا چند ویدئوی کوتاه کدهای تمرین را شرح دهید و به سوالات زیر به صورت کتبی پاسخ دهید.

۱ .شناسایی داده‌های پرت(Outliers)

- چرا برخی داده‌ها به عنوان داده پرت شناسایی می‌شوند؟ آیا حذف آن‌ها همیشه تصمیم درستی است؟

Outlierها به دادگانی اطلاق میشود که نسبت به داده‌های دیگر پرت باشند. بنا به دلایل مختلف مانند خطای اندازه‌گیری، اشتباهات و ... . اما حذف انها باعث ساده تر شدن و کمتر شدن خطای مدل شود. حذف داده‌های پرت باید با دقت و با توجه به تحلیل دقیق و هدف مورد نظر انجام شود. می‌توانید ابتدا تحلیل‌های آماری انجام داده و سپس تصمیم بگیرد.

- در چه شرایطی ممکن است داده‌های پرت اطلاعات مهمی درباره پدیده مورد مطالعه ارائه دهند؟

داده‌های پرت در شرایط غیرعادی، پدیده‌های نادر، نقص‌ها و ... ممکن است شامل داده‌های مفید باشند که حذف انها پیشنهاد نمیشود. در مواردی داده پرت ممکت است کلاس جداگانهو نادری باشد که در اندازه گیری شرایط اندازه گیری و ثبت انها در دیتاست وجود نداشته.

- مزایا و معایب استفاده از نمره Z در مقابل روش IQR برای شناسایی داده‌های پرت چیست؟

مزایای Z score :

- ✓ معیاری استاندارد: نمره Z به دلیل مقیاس‌بندی داده‌ها بر اساس میانگین و انحراف معیار، یک معیار استاندارد و قابل مقایسه در مقیاس‌های مختلف است.
- ✓ حساس به تغییرات: نمره Z نسبت به تغییرات کوچک در داده‌ها حساس است و می‌تواند داده‌های پرت را با دقت بالایی شناسایی کند.
- ✓ کاربرد در تحلیل‌های آماری: نمره Z به‌طور گسترده در تحلیل‌های آماری و مدل‌های پیش‌بینی استفاده می‌شود و می‌تواند اطلاعات دقیقی ارائه دهد.

معایب Z score :

- ❖ حساسیت به داده‌های پرت: نمره Z خود به داده‌های پرت حساس است و وجود داده‌های پرت ممکن است میانگین و انحراف معیار را به‌شدت تغییر دهد.
- ❖ فرضیه توزیع نرمال: استفاده از نمره Z فرض می‌کند که داده‌ها توزیع نرمال دارند. در صورت عدم توزیع نرمال داده‌ها، نتایج ممکن است غیر دقیق باشند.

مزایای IQR :

- ✓ مقاومت در برابر داده‌های پرت: روش IQR به دلیل استفاده از چارک‌ها، به داده‌های پرت حساس نیست و مقاوم‌تر در برابر آنها است.
- ✓ عدم نیاز به توزیع نرمال: روش IQR برای داده‌هایی که توزیع نرمال ندارند نیز قابل استفاده است و نتایج دقیقی ارائه می‌دهد.

معایب IQR :

- ❖ عدم مقیاس‌بندی استاندارد: روش IQR نسبت به نمره Z معیاری استاندارد ارائه نمی‌دهد و مقایسه داده‌ها در مقیاس‌های مختلف ممکن است دشوار باشد.
- ❖ حساسیت کمتر به تغییرات کوچک: روش IQR ممکن است داده‌های پرت کوچک‌تر را شناسایی نکند و نسبت به تغییرات کوچک حساسیت کمتری دارد.

- اگر مجموعه داده‌های شما شامل متغیرهای طبقه‌ای (دسته‌ای) باشد، چگونه می‌توان داده‌های پرت را شناسایی کرد؟

میتوانیم با تحلیل فراوانی، دسته ا تغداد کم را به عنوان داده پرت لحاظ کنیم. یا با روشهای clustering دسته های پرت را یافته و حذف کنیم.

- حذف داده‌های پرت چه تأثیری بر عملکرد یک مدل یادگیری ماشین دارد؟

حذف دادگان پرت میتواند به بهبود دقت مدل، کاهش نوسانات، بهبود کارایی و تعمیمپذیری داشته باشد. مثلا برای رگرسیون خطی داده پرت مدل دقیقی به ضورت خطی به ما نمیده و جبور به مدلی غیرخطی هستیم.

۲ .بررسی داده‌های از دست رفته(Missing Data)

- اگر یک مجموعه داده دارای مقادیر از دست رفته باشد، چگونه می‌توان تشخیص داد که این داده‌ها به صورت تصادفی از بین رفته‌اند یا الگوی خاصی دارند؟

روش بصری مانند باکس پلات هیستوگرتم و ... و یا missing value matrix . روش‌های اماری مانند ازمون فرض استقلال و یا تحلیل به وسیله مدل های ماشین لرنینگ برای یافتن ارتباط میان نبود داده.

- چه تأثیری می‌تواند حذف داده‌های ناقص بر روی مدل نهایی داشته باشد؟

بستگی به نوع داده و میزان استقلال نتیجه نهایی و داده‌های مورد نیاز ممکن است حذف انها خوب و یا بد باشد. مثلا میتواند به بهبود دقت مدل کمک کند، نویز و اغتشاش کاهش یابد، سرعت و کارایی افزایش یابد و یا از طرفی، منجر به از دست دادن داده های مهم، کاهش انداه نمونه و برهم خوردن توازن داده ها شود.

- کدام ستون‌های مجموعه داده تایتانیک بیشترین مقدار داده‌های گمشده را دارند؟ چرا فکر می‌کنید این داده‌ها گم شده‌اند؟

ستون شماره کابین و سن، که هرکدام میتواند به نحوه ثبت دادگان برگردد. مثلا بسته به کلاس مسافر، بعضی بلیت ها کابین مشخصی نداشته و بلیط های معمولی نیاز به دیتای خاصی برای سوار شدن نداشته‌اند و صرفا تهیه بلیط نیازمند پول بوده و نه داده هایی چون سن.

- دلایل احتمالی وجود داده‌های گمشده در مجموعه داده‌های دنیای واقعی چیست؟

خطای اندازه‌گیری، انسانی در جمع اوری داده، خرابی سیستم یا محدودیت منابع، ناتوانی در پاسخگویی و ..

۳ .حذف داده‌های ناقص(Removing Missing Data)

- در چه مواردی حذف داده‌های دارای مقدار از دست رفته روش مناسبی است؟

در حالتی که مقادیر از دست رفته کم باشد به نسبت کل دادگان.

- اگر درصد بالایی از یک ویژگی دارای مقادیر از دست رفته باشد، آیا حذف آن ویژگی منطقی است؟ چرا؟

بله. به این گونه پیچیدگی مدل کم شده، دقت افزایش میابد و مشکلات اماری کمتر میشوند/

- معایب احتمالی حذف سطرهایی که دارای مقادیر گمشده هستند چیست؟

اگر تعداد سطر به ستون کم باشد، حذف سطر به از دست رفتن داده و نمونه و تعمیم پذیری منجر میشود.

- اگر ۳۰٪ از داده‌های یک ستون گمشده باشد، آیا کل ستون را حذف می‌کنید؟ چرا یا چرا نه؟

ابتدا میبایست اهمیت ستون بررسی شود و تاثیر بر مدل ان. اگر اهمیت ستن کم باشد بله اما اگر تعداد ستون های داده کم باشد شاید تجدید نظر لازم باشد.

- چگونه می‌توان تصمیم گرفت که داده‌های گمشده را حذف کنیم یا مقدارگذاری(impute) کنیم؟

اگر درصد داده های گم شده باشد حذف بهتر است. بسته به نوغ داده، داده های عددی راحتتر impute میشود.

۴ .جایگزینی داده‌های از دست رفته(Imputation)

- چه تفاوتی بین جایگزینی داده‌های از دست رفته با میانگین، میانه و مد وجود دارد؟

جایگزینی با میانگین(Mean Imputation) : میانگین مقادیر موجود در یک ستون محاسبه شده و جایگزین مقادیر از دست رفته می‌شود.

مزایا: سادگی و سرعت: محاسبه و جایگزینی میانگین به سادگی و سرعت انجام می‌شود. حفظ توزیع کلی: در صورتی که داده‌ها به طور نرمال توزیع شده باشند، میانگین می‌تواند توزیع کلی داده‌ها را حفظ کند.

معایب: حساس به داده‌های پرت: میانگین به شدت تحت تأثیر داده‌های پرت قرار می‌گیرد. ایجاد نوسان مصنوعی: جایگزینی با میانگین ممکن است نوسانات مصنوعی در داده‌ها ایجاد کند و توزیع داده‌ها را تغییر دهد.

جایگزینی با میانه(Median Imputation) : میانه (میانگین مرکز) مقادیر موجود در یک ستون محاسبه شده و جایگزین مقادیر از دست رفته می‌شود.

مزایا: مقاومت در برابر داده‌های پرت: میانه تحت تأثیر داده‌های پرت قرار نمی‌گیرد و به همین دلیل مقاوم‌تر است. حفظ توزیع میانه: جایگزینی با میانه می‌تواند توزیع میانه داده‌ها را بهتر حفظ کند.

معایب: عدم حفظ جزئیات دقیق: میانه ممکن است نتواند جزئیات دقیق توزیع داده‌ها را حفظ کند.

جایگزینی با مد(Mode Imputation) : مد (بیشترین مقدار تکرار شده) مقادیر موجود در یک ستون محاسبه شده و جایگزین مقادیر از دست رفته می‌شود.

مزایا: مناسب برای داده‌های دسته‌بندی: مد به خوبی برای داده‌های دسته‌بندی (categorical) قابل استفاده است. حفظ پربسامدترین مقدار: جایگزینی با مد می‌تواند پربسامدترین مقدار داده‌ها را حفظ کند.

معایب: ایجاد سوگیری: جایگزینی با مد ممکن است باعث ایجاد سوگیری در داده‌ها شود و توزیع اصلی داده‌ها را تغییر دهد. عدم کاربرد برای داده‌های عددی: جایگزینی با مد ممکن است برای داده‌های عددی مناسب نباشد.

- در یک مسئله واقعی، چه استراتژی‌هایی برای جایگزینی داده‌های از دست رفته پیشنهاد می‌کنید؟

از روشهای مختلف imputation استفاده کنیم و یا روشهای اماری و یا روشهای خوشه بندی.

- در چه شرایطی بهتر است از مقدارگذاری میانگین به جای مقدارگذاری میانه استفاده شود؟

اگر توزیع داده ها نرمال باشد. اگر تاثیر داده های پرت قابل صرف نظر باشد. برای محاسبه سریع و کاهش پیچیدگی.

- چرا مقدارگذاری داده‌های گمشده بهتر از حذف آن‌ها است؟

چون اهمیت داده ممکن است به حدی باشد که حذف ان به سود ما نباشد. لذا وابستگی مدل به بعضی دادگان به قدری است که جایگذاری انها بهتر است.

- چگونه مقدارگذاری داده‌های گمشده می‌تواند باعث ایجاد سوگیری (Bias) در مجموعه داده شود؟

۵ .تقسیم داده‌ها به مجموعه آموزش و آزمایش(Train-Test Split)

- چرا تقسیم داده‌ها به دو مجموعه آموزش و آزمایش ضروری است؟
- اگر مجموعه آزمایش بسیار کوچک باشد، چه مشکلاتی ممکن است پیش بیاید؟
- در چه شرایطی از یک مجموعه اعتبارسنجی (Validation Set) علاوه بر مجموعه آزمایش استفاده می‌شود؟

۶. بصری سازی داده‌ها(Data Visualization)

- چگونه می‌توان با استفاده از روش‌های بصری‌سازی، روابط بین متغیرها را بهتر درک کرد؟
- در چه مواردی ممکن است تجسم داده‌ها منجر به تفسیرهای نادرست شود؟
- چه بینش‌هایی می‌توان از نمودار جفتی به دست آورد؟
- چگونه تجسم داده‌ها قبل از آموزش مدل می‌تواند به انتخاب ویژگی‌ها کمک کند؟
- اگر یک مجموعه داده توزیع متمایل داشته باشد، چه تکنیک‌های پیش‌پردازش می‌توان استفاده کرد؟

۷. بررسی تعادل کلاس‌ها در مجموعه داده(Data Balancing)

- چرا توزیع نامتعادل کلاس‌ها می‌تواند منجر به مشکلات در عملکرد مدل‌های یادگیری ماشین شود؟
- چه روش‌هایی برای متعادل‌سازی مجموعه داده پیشنهاد می‌کنید و مزایا و معایب هر روش چیست؟

۳- در فایل نوت بوک **exercise 1** که ضمیمه این تمرین هست جلوی برخی از کدها ##??????## قرار داده شده است. این دستورات را در اینترنت جستجو کرده و در خصوص آرگومان های ورودی که این کدها می توانند بگیرند، توضیح دهید.

```
sns.boxplot(x=df['petal length (cm)'])
```

column : ***str or list of str, optional***

Column name or list of names, or vector. Can be any valid input to `pandas.DataFrame.groupby()`.

by : ***str or array-like, optional***

Column in the DataFrame to `pandas.DataFrame.groupby()`. One box-plot will be done per value of columns in *by*.

ax : ***object of class matplotlib.axes.Axes, optional***

The matplotlib axes to be used by boxplot.

fontsize : ***float or str***

Tick label font size in points or as a string (e.g., *large*).

rot : ***float, default 0***

The rotation angle of labels (in degrees) with respect to the screen coordinate system.

grid : ***bool, default True***

Setting this to True will show the grid.

figsize : ***A tuple (width, height) in inches***

The size of the figure to create in matplotlib.

layout : ***tuple (rows, columns), optional***

For example, (3, 5) will display the subplots using 3 rows and 5 columns, starting from the top-left.

return_type : ***{'axes', 'dict', 'both'} or None, default 'axes'***

The kind of object to return. The default is `axes`.

- 'axes' returns the matplotlib axes the boxplot is drawn on.
- 'dict' returns a dictionary whose values are the matplotlib Lines of the boxplot.
- 'both' returns a namedtuple with the axes and dict.
- when grouping with `by`, a Series mapping columns to `return_type` is returned.

  If `return_type` is *None*, a NumPy array of axes with the same shape as `layout` is returned.

```
sns.heatmap(df.isnull(), cmap='viridis', cbar=False)
```

Parameters:

data : *rectangular dataset*

2D dataset that can be coerced into an ndarray. If a Pandas DataFrame is provided, the index/column information will be used to label the columns and rows.

vmin, vmax : *floats, optional*

Values to anchor the colormap, otherwise they are inferred from the data and other keyword arguments.

cmap : *matplotlib colormap name or object, or list of colors, optional*

The mapping from data values to color space. If not provided, the default will depend on whether `center` is set.

center : *float, optional*

The value at which to center the colormap when plotting divergent data. Using this parameter will change the default `cmap` if none is specified.

robust : *bool, optional*

If True and `vmin` or `vmax` are absent, the colormap range is computed with robust quantiles instead of the extreme values.

annot : *bool or rectangular dataset, optional*

If True, write the data value in each cell. If an array-like with the same shape as `data`, then use this to annotate the heatmap instead of the data. Note that DataFrames will match on position, not index.

fmt : *str, optional*

String formatting code to use when adding annotations.

annot_kws : *dict of key, value mappings, optional*

Keyword arguments for `matplotlib.axes.Axes.text()` when `annot` is True.

linewidths : *float, optional*

Width of the lines that will divide each cell.

linecolor : *color, optional*

Color of the lines that will divide each cell.

cbar : *bool, optional*

Whether to draw a colorbar.

cbar_kws : *dict of key, value mappings, optional*

Keyword arguments for `matplotlib.figure.Figure.colorbar()`.

cbar_ax : *matplotlib Axes, optional*

Axes in which to draw the colorbar, otherwise take space from the main Axes.

square : *bool, optional*

If True, set the Axes aspect to "equal" so each cell will be square-shaped.

xticklabels, yticklabels : *"auto", bool, list-like, or int, optional*

If True, plot the column names of the dataframe. If False, don't plot the column names. If list-like, plot these alternate labels as the xticklabels. If an integer, use the column names but plot only every n label. If "auto", try to densely plot non-overlapping labels.

mask : *bool array or DataFrame, optional*

If passed, data will not be shown in cells where `mask` is True. Cells with missing values are automatically masked.

ax : *matplotlib Axes, optional*

Axes in which to draw the plot, otherwise use the currently-active Axes.

kwargs : *other keyword arguments*

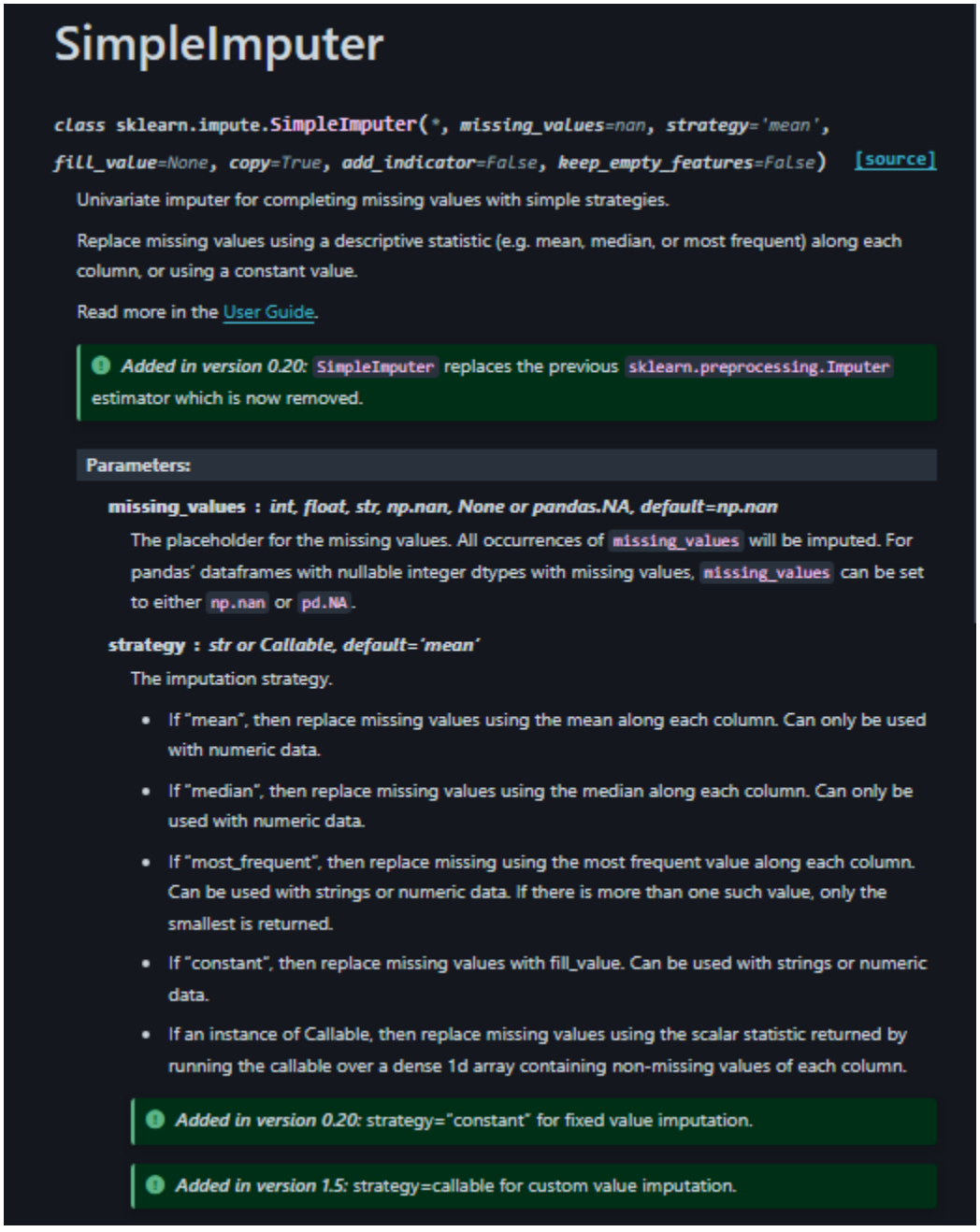
All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns:

ax : *matplotlib Axes*

Axes object with the heatmap.

```
num_imputer = SimpleImputer(strategy="mean")
```



SimpleImputer

class sklearn.impute.**SimpleImputer**(**, missing_values=nan, strategy='mean', fill_value=None, copy=True, add_indicator=False, keep_empty_features=False*) [source]

Univariate imputer for completing missing values with simple strategies.

Replace missing values using a descriptive statistic (e.g. mean, median, or most frequent) along each column, or using a constant value.

Read more in the User Guide.

Added in version 0.20: `SimpleImputer` replaces the previous `sklearn.preprocessing.Imputer` estimator which is now removed.

Parameters:

missing_values : *int, float, str, np.nan, None or pandas.NA, default=np.nan*

The placeholder for the missing values. All occurrences of `missing_values` will be imputed. For pandas' dataframes with nullable integer dtypes with missing values, `missing_values` can be set to either `np.nan` or `pd.NA`.

strategy : *str or Callable, default='mean'*

The imputation strategy.

- If "mean", then replace missing values using the mean along each column. Can only be used with numeric data.
- If "median", then replace missing values using the median along each column. Can only be used with numeric data.
- If "most_frequent", then replace missing using the most frequent value along each column. Can be used with strings or numeric data. If there is more than one such value, only the smallest is returned.
- If "constant", then replace missing values with fill_value. Can be used with strings or numeric data.
- If an instance of Callable, then replace missing values using the scalar statistic returned by running the callable over a dense 1d array containing non-missing values of each column.

Added in version 0.20: strategy="constant" for fixed value imputation.

Added in version 1.5: strategy=callable for custom value imputation.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

train_test_split

sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None) [source]

Split arrays or matrices into random train and test subsets.

Quick utility that wraps input validation, `next(ShuffleSplit().split(X, y))`, and application to input data into a single call for splitting (and optionally subsampling) data into a one-liner.

Read more in the User Guide.

Parameters:

arrays : sequence of indexables with same length / shape[0]

Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : float or int, default=None

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : float or int, default=None

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_state : int, RandomState instance or None, default=None

Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shuffle : bool, default=True

Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

stratify : array-like, default=None

If not None, data is split in a stratified fashion, using this as the class labels. Read more in the User Guide.

Returns:

splitting : list, length=2 * len(arrays)

List containing train-test split of inputs.

> *Added in version 0.16:* If the input is sparse, the output will be a `scipy.sparse.csr_matrix`. Else, output type is the same as the input type.

```
sns.countplot(x=df["survived"])
```

seaborn.countplot

`seaborn.countplot(*, x=None, y=None, hue=None, data=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, dodge=True, ax=None, **kwargs)`

Show the counts of observations in each categorical bin using bars.

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for `barplot()`, so you can compare counts across nested variables.

Input data can be passed in a variety of formats, including:

- Vectors of data represented as lists, numpy arrays, or pandas Series objects passed directly to the `x`, `y`, and/or `hue` parameters.
- A "long-form" DataFrame, in which case the `x`, `y`, and `hue` variables will determine how the data are plotted.
- A "wide-form" DataFrame, such that each numeric column will be plotted.
- An array or list of vectors.

In most cases, it is possible to use numpy or Python objects, but pandas objects are preferable because the associated names will be used to annotate the axes. Additionally, you can use Categorical types for the grouping variables to control the order of plot elements.

This function always treats one of the variables as categorical and draws data at ordinal positions (0, 1, ... n) on the relevant axis, even when the data has a numeric or date type.

See the tutorial for more information.

Parameters:

x, y, hue : *names of variables in `data` or vector data, optional*

Inputs for plotting long-form data. See examples for interpretation.

data : *DataFrame, array, or list of arrays, optional*

Dataset for plotting. If `x` and `y` are absent, this is interpreted as wide-form. Otherwise it is expected to be long-form.

order, hue_order : *lists of strings, optional*

Order to plot the categorical levels in, otherwise the levels are inferred from the data objects.

orient : *"v" | "h", optional*

Orientation of the plot (vertical or horizontal). This is usually inferred based on the type of the input variables, but it can be used to resolve ambiguity when both `x` and `y` are numeric or when plotting wide-form data.

color : *matplotlib color, optional*

Color for all of the elements, or seed for a gradient palette.

palette : *palette name, list, or dict*

Colors to use for the different levels of the `hue` variable. Should be something that can be interpreted by `color_palette()`, or a dictionary mapping hue levels to matplotlib colors.

saturation : *float, optional*

dodge : *bool, optional*

When hue nesting is used, whether elements should be shifted along the categorical axis.

ax : *matplotlib Axes, optional*

Axes object to draw the plot onto, otherwise uses the current Axes.

kwargs : *key, value mappings*

Other keyword arguments are passed through to `matplotlib.axes.Axes.bar()`.

Returns:

ax : *matplotlib Axes*

Returns the Axes object with the plot drawn onto it.

```
sns.pairplot(df[['age', 'fare', 'pclass', 'survived']], hue="survived")
```

seaborn.pairplot

```
seaborn.pairplot(data, *, hue=None, hue_order=None, palette=None, vars=None, x_vars=None,
y_vars=None, kind='scatter', diag_kind='auto', markers=None, height=2.5, aspect=1,
corner=False, dropna=False, plot_kws=None, diag_kws=None, grid_kws=None, size=None)
```

Plot pairwise relationships in a dataset.

By default, this function will create a grid of Axes such that each numeric variable in `data` will by shared across the y-axes across a single row and the x-axes across a single column. The diagonal plots are treated differently: a univariate distribution plot is drawn to show the marginal distribution of the data in each column.

It is also possible to show a subset of variables or plot different variables on the rows and columns.

This is a high-level interface for `PairGrid` that is intended to make it easy to draw a few common styles. You should use `PairGrid` directly if you need more flexibility.

Parameters:

data : `pandas.DataFrame`
Tidy (long-form) dataframe where each column is a variable and each row is an observation.

hue : *name of variable in* `data`
Variable in `data` to map plot aspects to different colors.

hue_order : *list of strings*
Order for the levels of the hue variable in the palette

palette : *dict or seaborn color palette*
Set of colors for mapping the `hue` variable. If a dict, keys should be values in the `hue` variable.

vars : *list of variable names*
Variables within `data` to use, otherwise use every column with a numeric datatype.

{x, y}_vars : *lists of variable names*
Variables within `data` to use separately for the rows and columns of the figure; i.e. to make a non-square plot.

kind : *{'scatter', 'kde', 'hist', 'reg'}*
Kind of plot to make.

diag_kind : *{'auto', 'hist', 'kde', None}*
Kind of plot for the diagonal subplots. If 'auto', choose based on whether or not `hue` is used.

markers : *single matplotlib marker code or list*
Either the marker to use for all scatterplot points or a list of markers with a length the same as the number of levels in the hue variable so that differently colored points will also have different scatterplot markers.

height : *scalar*
Height (in inches) of each facet.

aspect : *scalar*
Aspect * height gives the width (in inches) of each facet.

corner : *bool*
If True, don't add axes to the upper (off-diagonal) triangle of the grid, making this a "corner" plot.

dropna : *boolean*
Drop missing values from the data before plotting.

{plot, diag, grid}_kws : *dicts*
Dictionaries of keyword arguments. `plot_kws` are passed to the bivariate plotting function, `diag_kws` are passed to the univariate plotting function, and `grid_kws` are passed to the `PairGrid` constructor.

Returns:

grid : `PairGrid`
Returns the underlying `PairGrid` instance for further tweaking.